THE UNIVERSITY OF ALBERTA

PARALLEL ALGORITHMS FOR RELAXATION LABELINGS

by

Eva Kwan-sheung Leung

(C)

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF Master of Science

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL 1989

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR	: Eva Kwan-sheung Leung
TITLE OF THESIS	: Parallel Algorithms for Relaxation Labelings
DEGREE	: Master of Science
YEAR THIS DEGREE GRANTED	: Fall 1989

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purpose only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

Eva Leung

Permanent Address:
33 Maidstone Road,
9/F., To Kwa Wan,
Kowloon, Hong Kong.

Date: May 1, 1989

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled Parallel Algorithms for Relaxation Labeling submitted by Eva Leung in partial fulfilment of the requirements for the degree of Master of Science.

Xiaobo Li
(Supervisor)

W. A. Davis

J. Schaeffer

Date: May 1, 89

For Jesus Christ, My Lord and Savior

ABSTRACT

Relaxation labeling algorithms reduce uncertainties in object-label assignments using neighbor information. The algorithms are iterative and potentially parallel in updating the labeling in each iteration. Given N objects and M labels, the time complexity of the algorithms is $O(M^2N^2)$ per iteration on a sequential computer. This thesis derives a general computational model for relaxation labeling upon which parallel algorithms are developed for SIMD Illiac-mesh and Torus-mesh computers.

Researchers in the past decade have concentrated on developing parallel architectures and algorithms mainly for the discrete model. This thesis attempts to study other models as well. By considering a labeling as a matrix of dimension M by N and redefining the compatibility matrix to the same size, a general four-dimensional computational model is built including Rosenfeld *et al*four models and other non-linear probabilistic approaches. A table listing the notation and terminology used in the original formulations is included. Parallel algorithms are developed on SIMD machines based on the general computational model. Mesh algorithms using $M \times N$ PEs reduces the computational complexity to $O(MN)$. The algorithms are modified to work on less, say Z, PEs and the time complexity is $O\left\lceil \frac{M^2N^2}{Z} \right\rceil$.

Acknowledgements

Praise be to God for His unceasing love and guidance in every aspect of my life, without which the completion of the thesis would have been impossible.

The writing, rewriting and production of this thesis have been an arduous task, but one that has been made easier by the help of other people. I am especially grateful to my supervisor, Dr. Xiaobo Li, for his many invaluable advices which guide me through the production process. Thanks to the thesis examining committee members, Dr. F. Walter Bischof, Dr. Wayne Davis, Dr. Ahmed Kamal and Dr. Jonathan Schaeffer for their time and suggestions. Thanks also to Gordon Fong for helping me with many of the figures; Andy Nowacki and Andrew Leung for their proofreading.

I also wish to express my deepest gratitude to my family, especially Kent and Samson, for their encouragement and support. I am also indebted to my many friends: Sandie Hui, Helen Kwok, Shuk-mei Lai, Iris Lee, Joanna Liu, Esther Li, Stella Ma, Patricia Pang, So-ling & Stephen Tam, Sylvia Wong, Pastor & Mrs. Yu, brothers and sisters in the Paul Fellowship, who have supported me through their concern and countless prayers.

Funding assistance was provided by the Department of Computing Science, University of Alberta, and the National Science and Engineering Research Council of Canada.

Table of Contents

Chapter 1. Introduction 1

1.1. Introduction 1

1.2. Motivation for the research 8

1.3. Research Activities in Relaxation Labeling 9

1.3.1. Theoretical Aspect 9

1.3.2. Applications 10

1.4. Description of the thesis 11

Chapter 2. Generalized Formulations of Different Relaxation Labeling Approaches 12

2.1. Introduction 12

2.2. Original Formulations 13

2.3. Generalized Formulation 20

2.3.1. Defining a Labeling as a Matrix 20

2.3.2. Defining Compatibility Matrices 21

2.3.3. Defining Generalized Formulae 22

2.4 Conclusion 26

Chapter 3. Machine Model 28

3.1. Introduction 28

3.2. A Model of an Array Processor 28

3.3. Notations 31

3.4. Mesh Interconnection Network 31

3.5 Conclusion 33

Chapter 4. Parallel Algorithms and Performance Analysis on Relaxation Labeling 35

4.1. Introduction 35

4.2. Parallel Algorithms on SIMD Mesh Machine 36

4.2.1. Parallel Algorithms with MN PEs on Illiac Network 36
4.2.2. Parallel Algorithms with MN PEs on Torus Network 40
4.2.3. Parallel Algorithms with Less PEs 43
4.3. Performance Analysis 49
4.4 Conclusion 49
Chapter 5. Conclusion 52
6.1. Contributions of the Thesis 52
6.2. Further Work 53
References 54

List of Tables

Table 2.1. 13-20

(a) Notations and terminologies used in defining the discrete, fuzzy and linear probabilistic models by Rosenfeld *et al.* 13

(b) Notations and terminology definitions used by Rosenfeld *et al.*'s non-linear probabilistic model, Peleg's statistical approach, and Zucker and Mohammed's arithmetic updating formula. 15

(c) Notations and terminology definitions used by Hummel and Zucker in their discrete and weighted labeling models. 18

Table 2.2. Definition of functions in the general computational model for different labeling models. 23

Table 4.1. Performance of the mesh algorithms. 50

List of Figures

Fig. 1.1 Triangle Example. 3

(a) A triangle to be labeled. 3

(b-i) Unambiguous labelings of triangle. 3

(j) An illegal labeling. 3

Fig. 1.2 A consistent labeling which contains no unambiguous labeling. 5

Fig. 1.3 Decision tree for labeling the edges of a triangle. 5

Fig. 1.4 Relationship between the relaxation labeling and the general labeling problems. 7

Fig. 2.1 Viewing the relaxation labeling problem as a four-dimensional problem. 26

Fig. 3.1 A SIMD machine with an interconnection network. 29

Fig. 3.2 Examples of mesh. 34

(a) A 4×4 Illiac network. 34

(b) A 4×4 torus network. 34

Fig. 4.1 A M by N Illiac mesh. 38

Fig. 4.2 Contents of PE(λ, i). 38

Fig. 4.3 Storage of R in a 4×3 Torus mesh. 42

Fig. 4.4 Consecutive and cyclic storage of a matrix with $M=3, N=7, X=Y=2$. 45

(a) Original unpartitioned matrix. 45

(b) Consecutive storage. 45

(c) Cyclic storage. 45

Fig. 4.5 Processing and storage planes. 46

List of Notation

Notation	Description	Page
$U=\{u_1, \cdots, u_N\}$	A set of N objects	20
$\Lambda=\{\lambda_1, \cdots, \lambda_M\}$	A set of M labels	20
$l_i^{(k)}(\lambda)$	labeling of object u_i with label λ at k^{th} iteration	20
$L_i^{(k)}$	labeling vector for object u_i at k^{th} iteration	21
$L^{(k)}$	labeling matrix for all objects	21
$r_{ij}(\lambda, \lambda')$	compatibility between $u_i=\lambda$ and $u_j=\lambda'$	21
$R_{\lambda i}$	compatibility matrix of $u_{i=}\lambda$	22
w_{ij}	weighting factor specifying the importance of u_j to u_i	22
$g_1, \cdots, g_5$	generalized labeling operations (refer to Table 2.2)	22

CHAPTER 1
INTRODUCTION

1.1. INTRODUCTION

Given a set of unknown objects and a set of possible labels for the objects, the *general labeling problem* involves naming of the set of objects with the set of labels without violating some given *constraints*. The assignment of labels to objects is called a *labeling*. A labeling is called *unambiguous* if it assigns a single label to each object consistent with all constraints. A labeling is called *consistent* if it assigns a number of labels (one or more) to each object without violating any constraint. There may not exist a single label for each object satisfying all constraints at the same time.

The general labeling problem is a generalization of various problems, including the subgraph isomorphism problem, the graph homomorphism problem, the automata homomorphism problem, the graph coloring problem, the relational homomorphism problem, the packing problem, the scene labeling problem, the shape matching problem, the Latin square puzzle, constraint satisfaction problems, and theorem proving, etc. [HaS79]

Following the approach in [HaS79], the general labeling problem involves the following.

[1] A set of N objects, $U = \{u_1, \ldots, u_N\}$, for example, parts of an image to be labeled. An object is also called a *unit*.

[2] A set of M labels, $\Lambda = \{\lambda_1, \ldots, \lambda_M\}$, which are the possible names or interpretations for the objects.

[3] A compatibility model, also known as the world model, which is composed of :

[a] *The unit constraint relation*, $T \subseteq U^J$, is the set of all J-tuples of units which mutually constrain one another. Objects which mutually constraint one another are considered to be neighboring objects.

[b] *The unit-label constraint relation*, $R \subseteq (U \times \Lambda)^J$, is the set of all 2J-tuples $(u_{x_1}, \lambda_{y_1}, \ldots, u_{x_J}, \lambda_{y_J})$ where $(\lambda_{y_1}, \ldots, \lambda_{y_J})$ is one of the permitted or legal labelings of the units $(u_{x_1}, \ldots, u_{x_J})$.

Consider a simple scene analysis example taken from [RHZ76]. Given a triangle such as one in Fig. 1.1(a), we want to label each edge with one of the labels: $+, -, \rightarrow, \leftarrow$. The label "+" indicates a convex edge. The label "−" indicates a concave edge. The label "$\rightarrow$" indicates an occluding edge with the occluded surface on the left side of edge when following the triangle in a clockwise direction. The label "$\leftarrow$" indicates an occluding edge with the occluded surface on the left side of edge when following the triangle in an anti-clockwise direction. There are in total 4^3 or 64 ways to label the edges of the triangle. Out of these 64 labelings, only eight are physically possible and are shown in Fig. 1.1(b) - 1.1(i). Fig. 1.1(b) corresponds to a triangle floating above the background. Fig. 1.1(c) corresponds to a triangular hole in the background. Fig. 1.1(d)-(f) corresponds to triangular flaps folded upward. Fig. 1.1(g)-(i) corresponds to triangular flaps folded downward. Fig. 1.1(j) is an example of an impossible interpretation.

In the triangle example, each edge is a member of the object set, $U = \{u_1, u_2, u_3\}$ (Fig. 1.1(a)), and the set of possible labels is $\Lambda = \{+, -, \rightarrow, \leftarrow\}$. The unit-constraint relation defined over the object set is $T = \{(u_1, u_2), (u_2, u_3), (u_3, u_1)\}$. Each element in T is an unordered pair. The unit-label constraint relation, R, is the set $\{(u_1, +, u_2, \rightarrow), (u_2, -, u_3, \rightarrow), (u_3, \rightarrow, u_1, -), \ldots\}$. The set of all 64 labelings are also called possible labelings and all the labelings in Fig. 1.1(b)-(i) are called

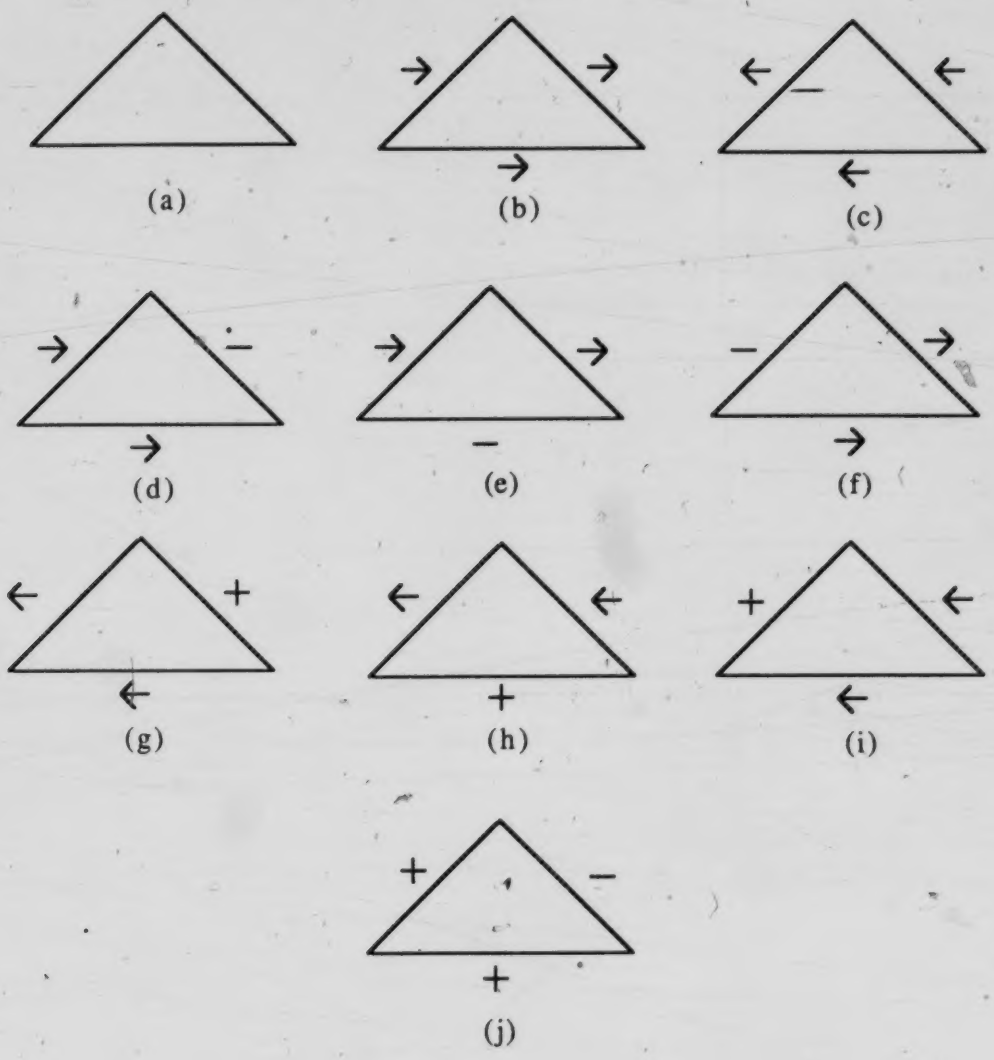


Fig. 1.1 Triangle Example
(a) A triangle to be labeled.
(b)-(i) Unambiguous labelings of the triangle.
(j) An illegal labeling.

unambiguous labelings.

A globally consistent labeling is the same as an unambiguous labeling. A locally consistent labeling associates a subset of labels to each object so that there always exists a label from the subset of each object that is consistent with the the set of constraints. However, there may not exist a single label for each of the objects in the whole object set satisfying all constraints at the same time.

Going back to the triangle, consider the following problem. Each edge of the triangle is an object in the object set, $U = \{u_1, u_2, u_3\}$. The tuples $(u_3, \rightarrow, u_1, -)$, $(u_1, \leftarrow, u_2, \leftarrow)$ and $(u_2, -, u_3, \rightarrow)$ belong to R. By assigning label subsets $\{\leftarrow, -\}$, $\{-, \leftarrow\}$ and $\{\rightarrow\}$ to objects u_1, u_2 and u_3 respectively, local consistency is achieved (Fig. 1.2). However, no globally consistent labeling can be derived from the above assignment.

One way to solve the labeling problem, that is, to find all unambiguous labelings, is to use depth-first search with backtracking as given in the procedure Search below.

```
Procedure Search (i, N)
Begin
    For each possible label for object u_i do
        Assign label to u_i
        If global consistency is attained then
            report unambiguous labeling
        EndIf
        Else
            Search (i+1, N)
        EndElse
        remove label for u_i
    EndFor
End
```

Fig. 1.3 shows a fragment of the decision tree generated by the procedure Search. The

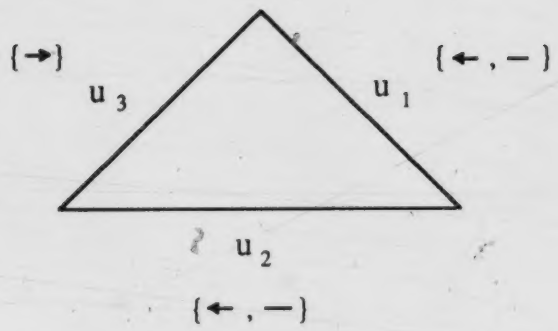


Fig. 1.2 A consistent labeling which contains no unambiguous labeling

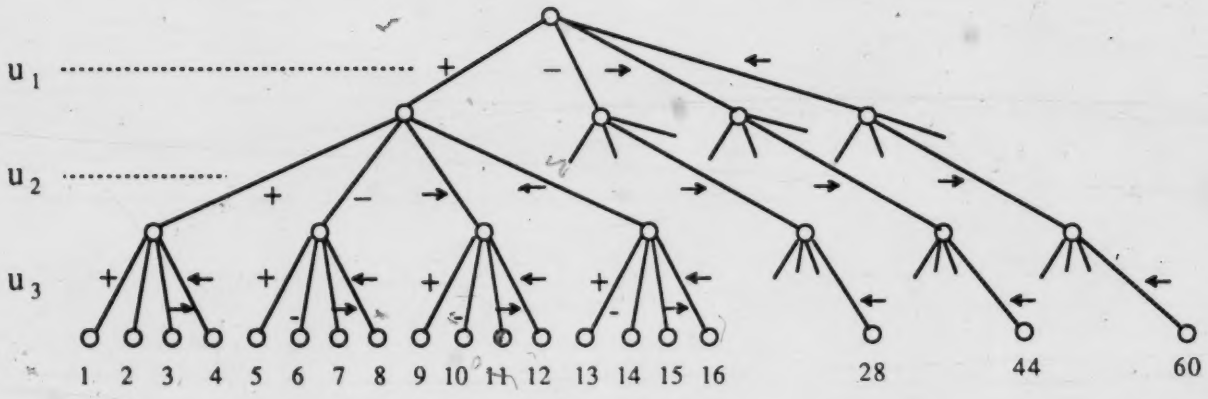


Fig. 1.3 Decision tree for labeling the edges of a triangle.

label for each of the edges u_1, u_2, u_3 is indicated on a branch of the tree. Path number 16 is an unambiguous labeling (u_1=+, u_2=←, u_3=←) corresponding to Fig. 1.1(g).

This labeling algorithm can be computationally inefficient. First, it does not prune the search tree efficiently. A poor choice of label for an unit will cause failure for all sequence paths resulting from that choice. Second, the algorithm re-examines paths that previously lead to failure. For example, in evaluating the path 12 (u_1=+, u_2=→, u_3=←) (Fig. 1.3), we find that the labels for u_2 and u_3 are inconsistent. However, the algorithm does not use the information later when evaluating the paths 28, 44 and 60, where the same label combination is tried out again.

Relaxation labeling approaches are thus introduced as a preliminary process to reduce the complexities in finding unambiguous labelings. Global consistency is attained by reducing local ambiguities using *contextual information*. In general, contextual information is defined on the degree of compatibilities between pairwise neighboring units which interact with the initial estimate of the labeling to give the final labeling. At each iteration, some identical operations are performed on each object to enable the propagation of local information for labeling modification. The goal is to find the greatest consistent labeling which is a consistent labeling with minimum number of labels in the labeling set of each object, and all unambiguous labelings for the problem can be derived from it. Relaxation labeling algorithms are all iterative in nature and are potentially parallel in each iteration.

The general labeling problem discussed so far is discrete in that it is either legal or illegal to assign a label to an object. This can be extended to the continuous domain when weights are associated with the assignments of labels to objects. In the fuzzy model, the weights are the degree of confidence in assigning a label to an object. In the linear and non-linear probabilistic models, the weights are probabilities or

correlations. Relationship between the development of the relaxation labeling and the general labeling problems is shown in Fig. 1.4.

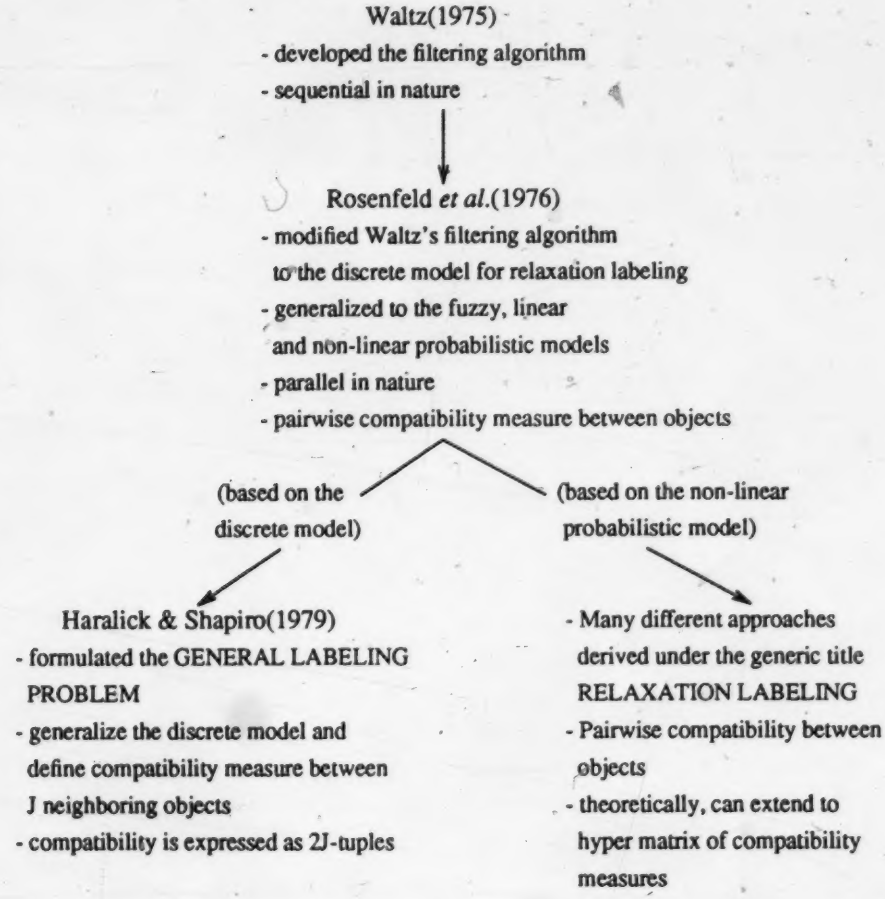

Waltz(1975)
- developed the filtering algorithm
- sequential in nature
Rosenfeld *et al.*(1976)
- modified Waltz's filtering algorithm to the discrete model for relaxation labeling
- generalized to the fuzzy, linear and non-linear probabilistic models
- parallel in nature
- pairwise compatibility measure between objects
(based on the discrete model)
(based on the non-linear probabilistic model)
Haralick & Shapiro(1979)
- formulated the GENERAL LABELING PROBLEM
- generalize the discrete model and define compatibility measure between J neighboring objects
- compatibility is expressed as 2J-tuples
- Many different approaches derived under the generic title RELAXATION LABELING
- Pairwise compatibility between objects
- theoretically, can extend to hyper matrix of compatibility measures

Fig. 1.4. Relationship between the relaxation labeling and the general labeling problems

1.2. MOTIVATION FOR THE RESEARCH

Originally, relaxation labeling algorithms were developed as a class of mechanisms served to deal with ambiguities and noises in vision systems. The general framework, however, has far broader potential applications and implications [HuZ83]. These techniques have been found to be powerful tools for reducing labeling errors in various types of image data and their performance is reported to be remarkably good in many domains. For instance, they are successfully applied to constraint propagation, particularly in "block-world" vision [Wal75]. In addition, relaxation labeling employed by simple tasks, such as labeling the sides of a triangle, has demonstrated that perfect labeling is possible [RHZ76, ZKH78]. As a result of their usefulness in various applications (refer to Section 1.3.2), a broad class of algorithms was developed and subsequently explored under the generic heading of relaxation labeling.

The attractiveness of relaxation labeling approaches is largely attributed to its ability to reduce the difficulties in attaining global consistency. This is achieved by the definition of a compatibility model between pairwise neighboring objects together with some simple local operations. Especially with the discrete model which can be applied to any problems posed as a labeling problem, interest in the algorithms has inspired applications in domains very different from computer visions.

Relaxation labeling algorithms are computationally intensive, especially with the probabilistic methods which are relatively slow when implemented on a conventional computer. An M-dimensional probability vector must be updated for each of the N objects in each iteration, and the algorithm has a time complexity of $O(M^2N^2)$ for one iteration on a sequential machine. Considerable speedup can be obtained if parallel machines are employed to implement the potentially parallel algorithms. In this thesis, we study the inherent computational properties of the problem, introduce

general formulae for different approaches, develop and analyze parallel algorithms on SIMD (Single Instruction Multiple Data) mesh machines with different number of processing elements (PEs).

1.3. RESEARCH ACTIVITIES IN RELAXATION LABELING

1.3.1. Theoretical Aspects

Different researchers have derived different relaxation algorithms and approaches attempting to tackle the labeling problem practically and analyze it theoretically. At first, numerous applications were developed without concrete underlying theoretical support. Being satisfied with successful applications, researchers generally found little motivation to further justify the approach and minimal efforts were made in analyzing the underlying equations. Attempts were made in later studies to develop a more solid theoretical foundation by introducing differential equations or a system of equations.

The two most commonly and successfully used algorithms were developed by Rosenfeld *et al.* [RHZ76] and Peleg [RPR81]. A total of four models are discussed in [RHZ76], namely, the discrete, the fuzzy, the linear and the non-linear probabilistic models. The non-linear probabilistic relaxation scheme uses correlations built on heuristics with limited theoretical support. Peleg employed a statistical approach based on Bayesian arguments. Several other algorithms have been proposed for modifying an initial labeling of a segmented scene using spatial relationship as contextual information. Many of them are actually variations of the relaxation techniques devised by [RHZ76]. Alternative developments have been made using different approaches. Faugeras and Berthod's optimization approach [FaB81] has used the

norm of difference between a label vector and an evidence vector as a definition consistency. The evidence vector is obtained from a computation involving each label's neighboring weights . Consistent labeling is obtained by minimization using standard methods. Hummel and Zucker's relaxation algorithm is based on variational calculus, which is an extension of the optimization viewpoint. Hinton developed relaxation labeling entirely within the constructs of linear programming [Hin79]. Recently, Thathachar and Sastry [ThS86] have proposed an alternative approach using learning automata interacting with an environment or a high-level critic that gives noisy response as to the consistency of a tentative labeling selected by the automata.

The convergence properties of the algorithms are discussed in [FER81, RLS81b, RHZ76, ZKH78]. See also [HuZ83] for a geometrical interpretation. Methods of accelerating the convergence are discussed in [RLS81a, ZKH78]. In most applications, the algorithm would not necessarily be iterated until a limit point is approached; rather, one would apply it for a few iterations and stop. Parallel algorithms and architectures, [DiM87, GW87, KTM88, LYD88, ReK85], are also developed mainly on Rosenfeld's discrete model.

1.3.2. Applications

Relaxation labeling algorithms are powerful tools commonly used in image processing and computer vision. Many different relaxation algorithms have been applied to various areas. Relaxation methods have been used in breaking substitution cipher [PeR79], pixel labeling [EYR78, LZR78], edge enhancement [HaR78, PeR78, RHZ76, SLZ77, Van76], line and curve enhancement [DaR80, DaR77, Zuc76, ZHR77], edge determination [Pra80], and scene analysis [BaT76]. Other applications include handwriting segmentation [Hay78, Pel79], shape

segmentation [RPR81], shape matching [Dav79], and clustering [ZuH77].

1.4. DESCRIPTION OF THE THESIS

Chapter 2 contains an overview of several relaxation labeling approaches with a table listing the notation and terminology used. Also included in this chapter are general formulations of the problem.

Chapter 3 describes a machine model upon which parallel algorithms are developed. Descriptions of the array processor machine model and the mesh interconnection network is given in this chapter.

Chapter 4 presents parallel algorithms developed on mesh interconnection network with different number of PEs and detailed performance analysis on the algorithms.

Chapter 5 is the conclusion of the thesis.

CHAPTER 2
GENERALIZED FORMULATIONS OF DIFFERENT RELAXATION LABELING APPROACHES

2.1. INTRODUCTION

This chapter states the the original updating rules employed by different relaxation labeling approaches upon which a general four-dimensional model is derived.

Among various discussions on relaxation labeling, similar notation and terminology have been used for different concepts. For instance, [RHZ76]'s definition of an unambiguous labeling is referred in [HuZ83] as a consistent labeling and the term unambiguous labeling is employed to describe any labeling which assigns only one label to each object. Sometimes, this creates confusion and misunderstanding in reading papers relevant to this subject. A table listing the notation and terminology used by some articles together with their original formulations are included in Table 2.1 of Section 2.2.

Three types of data structures have been used for a labeling: set, vector and matrix. The discrete and fuzzy labeling defined in [RHZ76] are N-tuples of subsets of labels. Most researchers [HuZ83, RHZ76, ThS86] defined a labeling as a MN-dimensional vector. Gu [GW87] defines a labeling as a matrix of dimension N by M. In this chapter, we develop a generalized formulation based on the matrix approach. The compatibility matrix is re-defined to develop the general model which formulates the discrete, fuzzy, linear and various non-linear probabilistic models. The general model does not intend to include every computational detail of every approach discussed. Only the central part of the updating technique is extracted into the model. In Section 2.3, we describe a general model for relaxation labeling algorithms including

Rosenfeld *et al*'s four models, Peleg's statistical formulation, Zucker and Mohammed's probabilistic approach, and Hummel and Zucker's supporting function. Section 2.4 contains the conclusion for the chapter.

2.2. ORIGINAL FORMULATIONS

Table 2.1a. Notations and terminologies used in defining the discrete, fuzzy and linear probabilistic models by Rosenfeld *et al.*

	Rosenfeld *et al.*'s [RHZ76]		
	Discrete	Fuzzy	Linear-Probabilistic
Approach	Iterative and potentially parallel version of Waltz filtering algorithm.	Generalized from the discrete model into the continuous domain. Define weighted labeling as confidence measures.	Weighted labeling model with labeling weights of each object sum to one. Employ linear updating rule.
Objects	$A = \{a_1, \ldots, a_n\}$. Individual object is denoted as a_i or a_j.	same	same
Labels	$\Lambda = \{\lambda_1, \ldots, \lambda_m\}$. Individual label is denoted as λ, λ' or μ.	$\Lambda = \{\lambda_1, \ldots, \lambda_m\}$. Individual label is denoted as λ, λ' or λ_j.	$\Lambda = \{\lambda_1, \ldots, \lambda_m\}$. Individual label is denoted as λ or λ'.
Arcs	—	—	—
Labeling of an object	A label set $L_i \subseteq \Lambda$ is assigned to each a_i, $1 \le i \le n$	A fuzzy subset L_i of Λ is assigned to each a_i, $1 \le i \le n$. L_i is a mapping from Λ into [0,1] which assigns a weight to each label. Weights of labels of an object need not sum to one.	$p_i(\lambda)$ is the label weight of λ at a_i where $0 \le i \le n$, $0 \le p_i(\lambda) \le 1$ and $\sum_{j=1}^{m} p_i(\lambda_j) = 1$. For each a_i, there corresponds a probability vector, $p_i = \left(p_i(\lambda_1), \ldots, p_i(\lambda_m)\right)$.
A labeling for all the objects	A labeling is an n-tuple of label sets: $L = (L_1, \ldots L_n)$.	A labeling is an n-tuple of fuzzy subsets: $L = (L_1, \ldots L_n)$.	A labeling is a mn-dimensional vector: $L^{(k)} = (p_1^{(k)}, \ldots p_n^{(k)})$

cont'

	Rosenfeld *et al.*'s [RHZ76]		
	Discrete	Fuzzy	Linear-Probabilistic
Unambiguous Labeling	An unambiguous labeling is consistent and assigns only a single label to each object.	same	same
Compatibility	Given $\Lambda_i \subseteq \Lambda$ to be the set of labels possible for a_i, $1 \le i \le n$. The set of compatible pairs of labels is Λ_{ij}. $(\lambda,\lambda') \in \Lambda_{ij}$ if it is possible to label a_i with λ and a_j with λ'.	A fuzzy label set, Λ_i, is a mapping from Λ into [0,1]. Λ_{ij} is the fuzzy set of pairs of labels. $\Lambda_{ij}(\lambda, \lambda')$ is the degree to which label λ of object a_i is compatible with label λ' of object a_j.	The compatibility factor, p_{ij}, is a mapping from $\Lambda \times \Lambda$ into [0,1] s.t. $\sum_{\lambda} p_{ij}(\lambda \mid \lambda') = 1$, for all i, j and λ'. $p_{ij}(\lambda \mid \lambda')$ is the conditional probability that object a_i has label λ given that object a_j has label λ'.
Consistency	The labeling, L, is consistent if for all i, j and $\lambda \in L_i$, we have $(\lambda \times L_j) \cap \Lambda_{ij} \neq \emptyset$.	The fuzzy labeling L is consistent if for all i, j and $\lambda \in L_i$, we have $\sup_{\lambda'} \left[L_j(\lambda') \wedge \Lambda_{ij}(\lambda,\lambda') \right] \ge L_i(\lambda)$, where $\wedge$ means "inf".	$p_i(\lambda)$ is consistent if for all i, j and $\lambda \in L_i$, we have $p_i(\lambda) = \sum_j c_{ij} \left[\sum_{\lambda'} p_{ij}(\lambda \mid \lambda') p_j(\lambda') \right]$ where coefficients, c's satisfy $\sum_j c_{ij} = 1$.
Updating algorithm	Discarding rule, Δ: To obtain the labeling at the $(k+1)$st step, we discard form each $L_i^{(k)}$ any label λ such that $(\lambda \times L_j) \cap \Lambda_{ij} \neq \emptyset$, for some j. The algorithm stops when $L^{(k+1)} = \Delta L^{(k)}$.	Label weakening algorithm, Ω: To obtain $L^{(k+1)}$ given $L^{(k)}$ we compute $L^{(k+1)}(\lambda) = \inf_j \left[\sup_{\lambda'} \left[L_j^{(k)}(\lambda') \wedge \Lambda_{ij}(\lambda,\lambda') \right] \right]$ for all i, j and λ. The algorithm stops when $L^{(k+1)} = \Omega L^{(k)}$.	Weight modification algorithm: To obtain $L^{(k+1)}$ from $L^{(k)}$ we compute $p_i^{(k+1)}(\lambda) = \sum_j c_{ij} \left[\sum_{\lambda'} p_{ij}(\lambda \mid \lambda') p_j^{(k)}(\lambda') \right]$ The algorithm converges to a unique final labeling depending only on compatibility between object pairs.

cont'

Table 2.1b. Notations and terminology definitions used by Rosenfeld *et al.*'s non-linear probabilistic model, Peleg's statistical approach, and Zucker and Mohammed's arithmetic updating formula.

	Rosenfeld *et al.* [RHZ76]	Peleg [Pel80]	Zucker *et al.* [ZuM78]
Approach	Probabilistic model based on heuristic.	Probabilistic relaxation using statistic approach based on Bayesian argument.	Probabilistic relaxation using arithmetic updating rule.
Objects	$A=\{a_1,\ldots,a_n\}$ is the set of object. Individual object is denoted as a_i or a_j.	Objects are denoted as a set of nodes, $V=\{v_1,\cdots,v_n\}$, in a graph G. Each node has J neighbors. Individual object is denoted as node i, j, v_i and v_j.	Objects are denoted as nodes of a graph and total number of nodes is I. Individual node is denotes as i or j.
Labels	$\Lambda=\{\lambda_1,\ldots,\lambda_m\}$ and individual label is denoted as λ, λ' and λ_i.	$\Lambda=\{\lambda_1,\ldots,\lambda_L\}$ is the set of labels. Individual label is denoted as α, β, λ, λ_1, λ_n, δ_1 or δ_n.	$\Lambda_i=\{\lambda_{i_1},\ldots\lambda_{i_L}\}$ is the set of labels associated with node i. Individual label is denoted as λ or λ'.
Arcs	—	$G=(V,E)$ is a graph and $E\subseteq V\times V$ is the set of arcs.	Arcs indicate neighboring pairs of objects.
Labeling of an object	$p_i(\lambda)$ is the label weight of λ at a_i where $0\le p_i(\lambda)\le 1$, $0\le i\le 1$, and $\sum_{j=1}^{m} p_i(\lambda)=1$. For each a_i, there corresponds a probability vector $p_i=\left[p_i(\lambda_1),\ldots,p_i(\lambda_n)\right]$	A probability vector, $P_{i_{(\alpha)}}:\Lambda\to[0,1]$, is an estimated discrete probability distribution for each random variable l_i associated with node v_i in the nth iteration. An iteration is sometimes denotes as k.	Attached to each label λ on each node i is a probabilistic measure of certainty, $p_i(\lambda)$ and $\sum_{\lambda\in\Lambda_i} p_i(\lambda_\alpha)$, $i=1,2,\ldots J$.
A labeling for all the objects	A labeling is an mn-vector, $L^{(k)}=(p_1^{(k)},\ldots,p_n^{(k)})$.	Not defined.	Not defined.

cont'

	Rosenfeld *et al.* [RHZ76]	Peleg [Pel80]	Zucker *et al.* [ZuM78]
Unambiguous Labeling	Not defined.	A labeling is unambiguous if every node has a determined label. The set of unambiguous labeling is also considered as a set of fixed points.	Not defined.
Compatibility	$r_{ij}(\lambda, \lambda')$ is the correlation between the events that a_i has label λ and a_j has label λ'. $r_{ij}(\lambda, \lambda')$ is positive if the two events frequently co-occur, negative is they rarely co-occur, and zero if their occurrences are independent. For $i=j$ we have $r_{ij}(\lambda, \lambda')=1$ for $\lambda=\lambda'$,and $=-p_i(\lambda)p_i(\lambda') / \sigma_i(\lambda)\sigma_i(\lambda')$ for $\lambda \neq \lambda'$.	Coefficients, $r_{ij}(\alpha,\beta)$'s, used in the updated formula which are related the occurrences between pairs of object label combinations. Given that $S_1, \ldots S_k$ and $A_1, \ldots A_n$ to be partitions of V and S resp. $r_{ij}(\alpha,\beta)$ is equal to $\frac{\dot{P}_m(l_i=\alpha, l_j=\beta)}{\dot{P}_k(l_i=\alpha)\cdot\dot{P}_l(l_j=\beta)}$ where $\dot{P}_k(l=\alpha) = \frac{1}{\|S_k\|}\sum_{s\in S_k} P_i^{(0)}(l_i=\alpha)$ and $\dot{P}_k(l_i=\alpha, l_j=\beta) = \frac{1}{\|A_k\|}\times \sum_{(v_i,v_j)\in A_k} P_i^{(0)}(l_i=\alpha)\cdot P_j^{(0)}(l_j=\beta)$.	$r_{ij}(\lambda,\lambda')$ in $[-1,1]$ indicate the compatibility that label λ on object i has with λ' on j. Both node and label dependencies are contained in the $r_{ij}(\lambda,\lambda')$. Conditional probability, $p_{ij}(\lambda\|\lambda')$ can be obtained by transformation. One such transformation is $p_{ij}(\lambda\|\lambda')=\frac{r_{ij}(\lambda\|\lambda')+1}{\alpha}$ where α is some suitable constant.
Consistency	Not defined.	No defined.	Not defined.

cont'

	Rosenfeld *et al.* [RHZ76]	Peleg [Pel80]	Zucker *et al.* [ZuM78]
Updating algorithm	Weight modifying algorithm: To obtain $p^{(k+1)}(\lambda)$ given $p^{(k)}(\lambda)$, the updating formulae are $q_i^{(k)}(\lambda) = \sum_j d_{ij} \left[\sum_{\lambda'} r_{ij}(\lambda,\lambda') p_j^{(k)}(\lambda') \right]$ and $p_i^{(k+1)}(\lambda) = \frac{p_i^{(k)}(\lambda)\left[1+q_i^{(k)}(\lambda)\right]}{\sum_{\lambda} p_i^{(k)}(\lambda)\left[1+q_i^{(k)}(\lambda)\right]}$	To obtain new estimated probability distributions, we compute the strength of support that label α at v_i gets from its neighbor v_j by $Q_{ij}(\alpha) = \sum_{\beta \in \Lambda} P_j^{(n)}(\beta) \cdot r_{ij}(\alpha,\beta)$. For all neighbors of i, we have $P_i^{(n+1)}(l_i=\alpha) = \frac{P_i^{(n)}(l_i=\alpha) \prod_{(i,j)\in E} Q_{ij}^{(n)}(\alpha)}{\sum_{\lambda} P_i^{(n)}(l_i=\lambda) \prod_{(i,j)\in E} Q_{ij}^{(n)}(\lambda)}$ It is shown that the algorithm will converge to some fixed point.	The update in each iteration involves the computation of neighborhood contribution to $p_i(\lambda)$ by $q_i^k(\lambda) = \sum_j C_{ij} \sum_{\lambda'} p_{ij}(\lambda \mid \lambda') p_j^k(\lambda')$, where C_{ij} indicate the proportional influence that node j can exert on node i, and $\sum_{j=1}^{J} C_{ij} = 1$. Then update it by $p_i^{(k+1)}(\lambda) = \frac{p_i^k(\lambda) q_i^k(\lambda)}{\sum_{\lambda'} p_i^k(\lambda') q_i^k(\lambda')}$

cont'

Table 2.1c. Notations and terminology definitions used by Hummel and Zucker in their discrete and weighted labeling models.

	Hummel *et al.* [HuZ83]	
	(discrete)	(weighted)
Approach	Discrete relaxation based on variational calculus.	Probabilistic model based on variational calculus.
Objects	An object is a node of a graph denoted by its index, i, where $1 \le i \le n$.	An object is a node of a graph denoted by its index, i, where $1 \le i \le n$.
Labels	Λ_i is a set of labels attached to node i. Individual label is denote as λ, λ' and λ''.	Λ_i is a set of labels attached to node i. Individual labels as denoted λ, λ' and λ''.
Arcs	—	—
Labeling of an object	$p_i(\lambda)$ indicates whether λ is associated with node i, according to $p_i(\lambda) = \begin{cases} 1, \lambda \text{ is label of } i, \\ 0, \text{otherwise.} \end{cases}$	$p_i(\lambda)$ indicates the weight of associating label λ with node i and $1 \le p_i(\lambda) \le 1$ and $\sum_{\lambda=1}^{m} p_i(\lambda) = 1$.
A labeling for all the objects	A labeling is an mn-vector, $\bar{p} = (\bar{p}_1, \ldots \bar{p}_n)$.	A labeling is an mn-vector in the space of weighted labeling assignments defined as: $K = \{\bar{p} \in R^{mn}: \bar{p} = (p_1(1), \ldots p_i(m)) \in R^m; 0 \le p_i(\lambda) \le 1, \text{ all } i, \lambda; \sum_{\lambda=1}^{m} p_i(\lambda) = 1 \text{ for } i = 1, \ldots, n \}$.
Unambiguous Labeling	An unambiguous labeling assignment is a mapping from the set of objects into the set of all labels, so that each object is associated with exactly one label.	An extension to the concept of an unambiguous labeling assignment is the concept of weighted labeling assignment in which the condition $p_i(\lambda) = 0$ or 1 is replaced by the condition $0 \le p_i(\lambda) \le 1$.

cont'

	Hummel *et al.* [HuZ83]	
	(discrete)	(weighted)
Compatibility	$R_{ij}(\lambda,\lambda') = \begin{cases} 1, \text{ if } (\lambda,\lambda') \in \Lambda_{ij}, \\ 0, \text{ if } (\lambda,\lambda') \in \Lambda_{ij}. \end{cases}$ represent the constraints.	The compatibility matrix contains real numbers obtained by compatibility function, $r_{ij}(\lambda,\lambda')$ in $(-\infty,+\infty)$, with its positivity proportional to the extend of local consistency between assignments, (i,λ) and (j,λ'), and its negativity to inconsistency. It is zero if no interaction between assignments. The compatibility matrix can generalized to be a hyper-dimensional matrix in which compatibility is defined on more than two assignments.
Consistency	Define the support for label λ at object i by the labeling assignment $\bar{p}$ (unambiguous or not) is: $s_i(\lambda) = s_i(\lambda;\bar{p}) = \sum_{j=1}^{n}\sum_{\lambda'=1}^{m} r_{ij}(\lambda,\lambda')p_j(\lambda')$. The unambiguous labeling $\bar{p}$ is consistent providing that it satisfies a system of inequalities: $\sum_{\lambda=1}^{m} p_i(\lambda)s_i(\lambda;\bar{p}) \geq \sum_{\lambda=1}^{m} v_i(\lambda)s_i(\lambda;\bar{p}),\ i=1,\ldots,n,$ for all unambiguous labelings $\bar{v}$.	Define the support for label λ at object i by the weighted labeling assignment $\bar{p}$ is: $s_i(\lambda) = s_i(\lambda;\bar{p}) = \sum_{j=1}^{n}\sum_{\lambda'=1}^{m} r_{ij}(\lambda,\lambda')p_j(\lambda')$. Then $\bar{p}$ is consistent providing that it satisfies a system of inequalities: $\sum_{\lambda=1}^{m} p_i(\lambda)s_i(\lambda;\bar{p}) \geq \sum_{\lambda=1}^{m} v_i(\lambda)s_i(\lambda;\bar{p}),\ i=1,\ldots,n,$ for all weighted labelings assignment $\bar{v}$.

cont'

	Hummel *et al.* [HuZ83]	
	(discrete)	(weighted)
Updating algorithm	For each iteration: (1) compute $\bar{q}^k$ = $\sum_j \sum_{(\lambda')} r_{ij}(\lambda,\lambda') p_j(\lambda')$. (2) set c_i = $\frac{1}{m}\sum_{l=1}^{m} q_i(l) \qquad i = 1, \ldots, n$ (3) set all $w_i(\lambda) = q_i(\lambda) - c_i$. (4) set $u_i(\lambda) = \frac{w_i(\lambda)}{\|\bar{w}\|}$, all i, λ. where $\|\bar{w}\| = \left[\sum_{i,\lambda} w_i(\lambda)^2\right]^{1/2}$ (5) if $\bar{u}^k = 0$, stop (6) set $\bar{p}^{k+1} = \bar{p}^k + h\bar{u}^k$, where $0 < h \le \alpha_k$ is determined so that $\bar{p}^{k+1} \in K$. α_k is some predetermined small value.	

2.3. GENERALIZED FORMULATION

2.3.1. Defining a Labeling as a Matrix

Consider a set of N objects, $U = \{u_1, \cdots, u_N\}$ and a set of M labels, $\Lambda = \{\lambda_1, \cdots, \lambda_M\}$. A mapping, $l_i : \Lambda \rightarrow Range$, defines a labeling on object u_i where the range of l_i, *Range*, depends on the particular model employed. In the discrete model, *Range* is the set $\{0,1\}$, and $l_i(\lambda)=1$ indicates that λ is a legal (allowed) label for object u_i, while $l_i(\lambda)=0$ means that it is impossible to label u_i with λ. In the fuzzy, linear and non-linear models, *Range* is the closed interval $[0,1]$; and $l_i(\lambda)$ represents the degree of confidence or the probability that u_i being assigned label λ. The probability can be defined statistically as conditional probability or correlation, etc. We use superscripts to denote iterations. The notation $l_i^{(k)}(\lambda)$ represents our belief about

labeling u_i with λ at the k-th iteration, while k=0 is the initial labeling restriction obtained from a priori knowledge. Square brackets are sometimes used to denote an element of a matrix.

The labeling vector $L_i^{(k)}$ for object u_i is defined below:

$$L_i^{(k)} = \begin{bmatrix} l_i^{(k)}(\lambda_1) \\ l_i^{(k)}(\lambda_2) \\ \vdots \\ l_i^{(k)}(\lambda_M) \end{bmatrix}_{M\times 1}$$

We will first consider that each unit has N neighbors. Define $L^{(k)} = [L_1^{(k)}\ L_2^{(k)} \ldots L_N^{(k)}]$ to be a M by N labeling matrix so that $L^{(k)}[\lambda, i] = l_i^{(k)}(\lambda)$.

2.3.2. *Defining Compatibility Matrices*

Another mapping, $r_{ij}: \Lambda\times\Lambda\rightarrow Range$, defines the effect of object u_j on object u_i. The range of r_{ij}, *Range*, is dependent on the particular model employed. We use u_i=λ to represents the assignment of label λ to object u_i. The meaning of $r_{ij}(\lambda, \lambda')$ in each model is listed below:

In the discrete model,

$r_{ij}(\lambda, \lambda')$=1: u_i=λ is compatible with u_j=λ';

$r_{ij}(\lambda, \lambda')$=0: u_i=λ is not compatible with u_j=λ'.

In the fuzzy model,

$r_{ij}(\lambda, \lambda')$ is the confidence of assigning λ to u_i

given the assignment of λ' to u_j.

In the linear and non-linear probabilistic model,

$r_{ij}(\lambda, \lambda')$ is the probability of assigning λ to u_i given the assignment of λ' to u_j.

The compatibility matrix of unit i with λ is defined as

$$R_{\lambda i} = \begin{bmatrix} w_{i1} \cdot r_{i1}(\lambda, \lambda_1) & w_{i2} \cdot r_{i2}(\lambda, \lambda_1) & \dots & w_{iN} \cdot r_{iN}(\lambda, \lambda_1) \\ w_{i1} \cdot r_{i1}(\lambda, \lambda_2) & w_{i2} \cdot r_{i2}(\lambda, \lambda_2) & \dots & w_{iN} \cdot r_{iN}(\lambda, \lambda_2) \\ \vdots & \vdots & \vdots & \vdots \\ w_{i1} \cdot r_{i1}(\lambda, \lambda_M) & w_{i2} \cdot r_{i2}(\lambda, \lambda_M) & \dots & w_{iN} \cdot r_{iN}(\lambda, \lambda_M) \end{bmatrix}_{M \times N}$$

Element $R_{\lambda i}[\lambda', j]$ corresponds to $w_{ij} \cdot r_{ij}(\lambda, \lambda')$. where w_{ij} is a weighting factor specifying the importance of u_j to u_i regardless of the particular labels involved. Since the values of $r_{ij}(\lambda, \lambda')$ and w_{ij} remain unchanged from iteration to iteration, we include the w_{ij} into the definition of $R_{\lambda i}$ so that the multiplication needs not be done repeatedly for each iteration.

2.3.3. *Defining Generalized Formulae*

We now proceed to define the relaxation operation for discrete, fuzzy and linear probabilistic models as an operation on the labeling matrix L_i and the compatibility matrix $R_{\lambda i}$. In the fuzzy model, the label weakening algorithm in its original notation is:

$$L^{(k+1)}(\lambda) = \inf_j \left[\sup_{\lambda'} \left[L_j^{(k)}(\lambda') \wedge \Lambda_{ij}(\lambda, \lambda') \right] \right] \qquad (1)$$

for all i, j and λ (refer to updating algorithm for the fuzzy model in Table 2.1a). Defining $g_3 = \min_j$, $g_2 = \max_{\lambda'}$ and the binary operation $g_1 = \min$, equation (1) can be written in our notation as

$$L^{(k+1)}[\lambda, i] = g_3 g_2 \bar{g}_1 (L^{(k)}, R_{\lambda i}).$$

In a similar way, we can define the discrete and the linear probabilistic relaxation operations, as shown in Table 2.2. Table 2.2 gives the definitions of $\bar{g}_1$, g_2, g_3 and other functions which will be used for the non-linear model.

Table 2.2. Definition of functions in the general computational model for different labeling models.

model	g_5	$\bar{g}_4$	g_3	g_2	$\bar{g}_1$
discrete	-	-	$\min_j$	$\max_{\lambda'}$	min
fuzzy	-	-	$\min_j$	$\max_{\lambda'}$	min
linear probabilistic	-	-	$\sum_j$	$\sum_{\lambda'}$	×
non-linear probabilistic					
Rosenfeld *et al.*	$\sum_{\lambda''}$	×	$\sum_j$	$\sum_{\lambda'}$	×
Hummel *et al.*	-	-	$\sum_j$	$\sum_{\lambda'}$	×
Peleg	$\sum_{\lambda''}$	×	$\prod_j$	$\sum_{\lambda'}$	×
Zucker *et al.*	$\sum_{\lambda''}$	×	$\sum_j$	$\sum_{\lambda'}$	×

Notice that $\bar{g}_1$ is a binary operator on corresponding elements of the two matrices of

the same dimension and returns a resultant matrix of the same size. We denote $G_3 = g_3 g_2 g_1$. The G_3 function is a very common operation among various relaxation labeling approaches. Not only has it been applied to Rosenfeld *et al.*'s three models, it is also used in the non-linear approaches to obtain neighboring effect on an object. The support function of Hummel *et al.*, the non-linear probabilistic model of Rosenfeld *et al.*, and the non-linear updating formula of Zucker *et al* all involve the computation of

$$q_i(\lambda) = \sum_j \sum_{\lambda'} \left[r_{ij}(\lambda, \lambda'), l_j(\lambda') \right]$$

Peleg used the following calculation in his statistical model,

$$q'_i(\lambda) = \prod_j \sum_{\lambda'} \left[r_{ij}(\lambda, \lambda'), l_j(\lambda') \right]$$

Both equations can be formulated as G_3 composite function.

Similarly, we can define a generalized formula for various non-linear probabilistic updating rules. In the non-linear probabilistic model, the weight modifying algorithm in its original notation is defined as follows:

$$q_i^{(k)}(\lambda) = \sum_j d_{ij} \left[\sum_{\lambda'} r_{ij}(\lambda,\lambda') p_j^{(k)}(\lambda') \right]$$

and

$$p_i^{(k+1)}(\lambda) = \frac{p_i^{(k)}(\lambda) \left[1+q_i^{(k)}(\lambda) \right]}{\sum_{\lambda} p_i^{(k)}(\lambda) \left[1+q_i^{(k)}(\lambda) \right]} \qquad (2)$$

(refer to the updating algorithm of Rosenfeld *et al.* in Table 2.1b). Defining $g_5 = \sum_{\lambda''}$,

$g_4 = \times$, $g_3 = \sum_j$, $g_2 = \sum_{\lambda'}$ and $g_1 = \times$, equation (2) can be written as

$$L[\lambda, i] = \frac{g_4\left[L[\lambda, i], G_3(L, R_{\lambda i})\right]}{\underset{\lambda''}{g_5}\, g_4\left[L[\lambda'', i], G_3(L, R_{\lambda'' i})\right]}$$

In a similar way, we can define the other non-linear probabilistic relaxation operations as shown in Table 2.2. Again, g_1 and g_4 operate on two operands while the rest operate on multi-operands.

So far we have considered N neighbors for each object. Generalization to J objects and W possible labels, where $J \leq N$ and $W \leq M$, for each object can easily be accommodated. W is the number of possible labels for u_i, denoted as Λ_i. Assuming that each unknown object has exactly J neighbors, if $i_1, \ldots, i_J$ are J integers then $\{u_{i_1}, \ldots, u_{i_J}\}$ is the set of the J neighbors of u_i. Define $N_i^{(k)} = [L_{i_1}, \ldots, L_{i_J}]$ be the M by J neighboring matrix of unit u_i at the k-th iteration with L_{i_1} be the M-dimensional labeling (row) vector of u_{i_1}. Similarly, if we consider only the subset Λ_i, then the dimension of the matrix $N_i^{(k)}$ becomes W by J. In the same way, we can re-define $R_{\lambda i}$ to dimension W by J and apply the g_i's to the newly defined $R_{\lambda i}$ and $N_i^{(k)}$ matrices which are smaller in size.

The relaxation labeling problem can be viewed as a four-dimensional problem (Fig. 2.1). The general model involves three multi-operand (or "accumulating") operators; each operates along a different dimension, namely, j, λ and λ'. In relation to a given object, the three dimensions j, λ and λ' represent its neighbors, possible labels for itself and possible labels for its neighbors respectively. The fourth direction is the dimension for the propagating of labeling information throughout the model ready for updating in the next iteration. Matrix $R_{\lambda i}$ is specified on the j and λ' plane

fixing both i and λ while $R_{ij}(\lambda, \lambda')$ is specified on another plane. Our general model conducts an update via the following path. To update $l_i(\lambda)$, the $l_j(\lambda)$'s are collected along the i axis and G_3 operation is applied to the j-λ' plane. The resulting effect on the $l_i(\lambda)$ is obtained by applying $\bar{g}_4$ and then normalized along the λ axis.

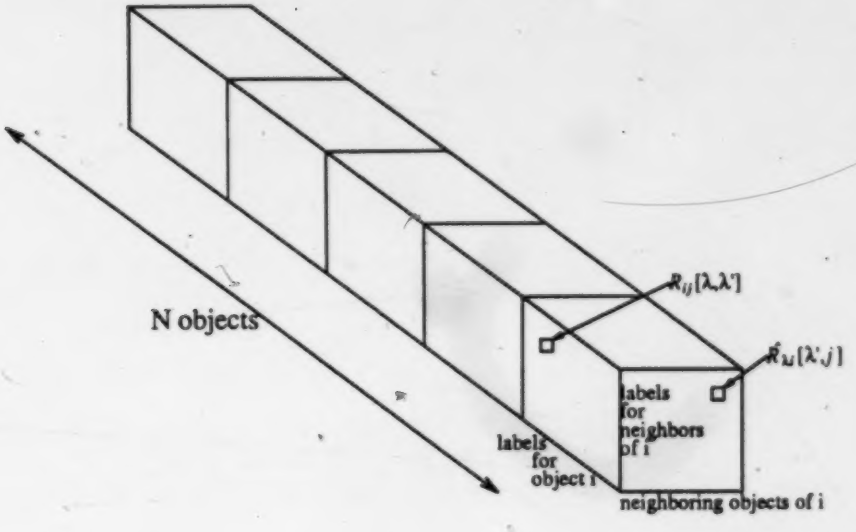


Fig. 2.1. Viewing the relaxation labeling problem as a four dimensional problem

2.4. CONCLUSION

In this chapter, we developed a general computational model for several well-known relaxation labeling algorithms. By considering a labeling as a matrix of dimension M by N, a four-dimensional general computational model for relaxation labeling is built based the non-linear probabilistic approaches by Rosenfeld *et al.*,

Peleg, Zucker *et al.* and the support function by Hummel *et al.* In the discussion of the original formulations of different approaches, different notations and terminologies have been employed by different researchers. A table containing definitions of notations and terminologies is included to list the original formulations of different approaches. No normalizing is needed for the discrete, fuzzy and linear-probabilistic model, therefore, they can be defined as a three-dimensional problem using the data organization of the general model. The four dimensions of the problem are denoted as j, λ', i, λ which represent the axes of object's neighbors, neighbor's labels, different objects, and an object's possible labels, respectively. The compatibility matrix is defined as a two-dimensional array of compatibility matrices between a given object and a possible label for the object. Each iteration involves the mapping of a pile or a three-dimensional array of compatibility matrices, $R_{\lambda i}$, for $\lambda = 1, \ldots M$, with the labeling matrix, L, to give an updated scalar, $l_i(\lambda)$. Updated $l_i(\lambda)$ are then distributed to along the fourth dimension to different cubes for updating in the next iteration. Operations are defined as five g functions which maybe different for different approaches. This general model provides the framework for the development of general parallel relaxation labeling algorithms which can easily be modified to suit individual needs of different applications.

CHAPTER 3
MACHINE MODEL

3.1. INTRODUCTION

One characteristic feature in the task of relaxation object labeling is that it involves many identical operations performed on all the units simultaneously during each iteration. Results obtained in each iteration are determined based on those in the previous iteration. Thus, the computation involved can readily be vectorized. The SIMD array processor which handles single instruction and multiple data streams is a natural computer architecture for such task.

In Section 3.2, we present the general framework of a SIMD array processor. Notations for parallel algorithms are listed in Section 3.3. In Section 3.4, we describe the mesh network in detail. A short summary is presented in Section 3.5.

3.2. A MODEL OF AN ARRAY PROCESSOR

An array processor is composed of a number of identical processors, the processing elements (PEs). The PEs are synchronized and execute instructions issued from a control unit (CU). The CU broadcasts an instruction to all PEs, and all enabled PEs simultaneously execute the instruction. The enable/disable mask can be used to select a subset of PEs to perform an instruction; and the mask can be changed from instruction to instruction. No synchronization primitives are needed during computation. Parallel algorithms on SIMD machines pre-schedule all PEs on the systems. A interconnection network is used to provide inter-PE communications. It should be emphasized that, in any time instance, only one unit of data can be transmitted along

an interconnection line, although all lines can be busy at the same time. Figure 3.1 taken from [HwB84] depicts the general machine model used in the thesis.

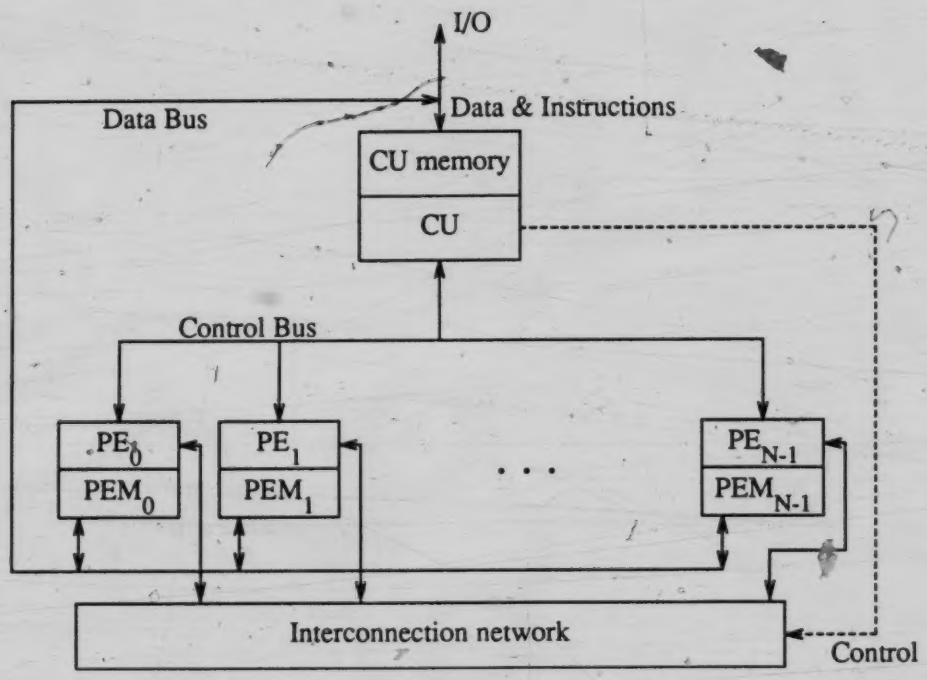


Figure 3.1. A SIMD machine with an interconnection network

In our model, the array processor is composed of $Q = 2^q$, for some integer q, PEs indexed 0 through $Q-1$. Each has a local memory of size $O(MN)$ and several registers. Four registers are dedicated for special functions:

[1] Memory is addressed by setting the appropriate values in the memory address register (MAR) in each PE.

[2] The address of each PE, the PE index, is stored in the the PE's address register (PAR).

[3] The transfer of register content from one PE to another actually consists of three steps. First, the register content is transferred from the source register to the routing register of the sending PE. The content is then transferred from the routing register of the sending PE to the routing register of the receiving PE via an interconnection network. Finally, the content is transferred from the routing register of the receiving PE to the destinated register. All these steps are handled by the PEs automatically in "one" transfer step. Some array processor may use two registers, one for input and the other for output. Others may use just one register in which the inputs and the outputs of the register are totally isolated by using master-slave flip-flops. Hence, each PE can send and receive data at the same time within "one" transfer step.

[4] A PE is either active or inactive as controlled by the value set in its mask register. Conventionally, a one is chosen for an active PE and 0 for an inactive one.

All other registers are prefixed with the letter R. Register $R0$ in PE(p) is referred to as $R0(p)$. For convenience, other names are also used to denote the registers. When the meaning is obvious from the context, the index p is omitted for simplicity. If a register name is used in a statement without any index, it means that this operation is executed for all PEs.

3.3. NOTATIONS

In describing our algorithms, we follow the notation common in the literature [DNS81, FLN85]. The symbol "←" denotes an assignment involving data routing between directly connected PEs; ":=" denotes local assignment in one PE; while "⇐" denotes an assignment requiring common data or constant movement from the control unit memory to PE registers. For example, if we want to transfer information from registers in PEs with odd row numbers up to those with even row numbers, the statement will be

$$R0((i-1) \bmod Q, j) \leftarrow R0(i, j)(i_0 = 1)$$

The condition in parentheses after the assignment is the enable mask. Consider $i = i_{n-1} i_{n-2} \ldots i_0$ to be the binary representation of the row index with i_{n-1} to be the most significant bit (MSB) and i_0 the least significant bit (LSB). If the LSB of the row index of the PE is 1, it is enabled; otherwise, disabled. If no mask function is provided, all PEs are enabled. The "mod" function is used to ensure wraparound transfers. It will be assumed and omitted for simplicity on further discussions. Thus, a typical statement will be as follows:

$$R0(i-1, j) \leftarrow R0(i, j) \quad (i_0=1)$$

3.4. MESH INTERCONNECTION NETWORK

The PEs maybe thought of being physically arranged in a k-D array $Q = Q_k \times Q_{k-1} \times \cdots \times Q_1$, where Q_i is the size of the ith dimension. The PE$(p_{k-1}, \ldots, p_0)$ is connected to PE$(p_{k-1}, \ldots, p_x \pm 1, \ldots, p_0)$, $0 \le x \le k$, provided that they exist. Each PE has at most $2k$ links. Data may be transmitted from one PE to

another via this interconnection pattern only. We consider the interconnection scheme for k=2. The PEs form a 2-D array of size $Q_1 \times Q_2$, for some integer Q_1, Q_2.

The mesh interconnection network is the most popular network for SIMD machines. Each PE can be considered as connected to its four immediate neighbors (above, below, left and right). Different versions of mesh network have different setup for the connection of the boundary PEs, and two of them will be considered, namely, the boundary wraparound mesh and the boundary end-around mesh. A typical example of the boundary wraparound mesh is the Illiac Network; so it is also known as an Illiac Network. In an Illiac Network, each PE at one end of the array is connected to another PE at the opposite end of the next row, and similarly for column PE connections. Hence, sometimes it can be viewed as having all PEs connected in a 1-D array in a circular fashion [HwB84] along the rows or the columns. In the boundary end-around mesh, also known as Torus Network, each PE at one end of the array is connected to another PE at the opposite end of the same row, and similarly for column PE connections. Fig. 3.2a and 3.2b shows the configurations of an Illiac Network and a torus respectively. Some machines, such as NASA's MPP are capable of being connected either as a torus or an Illiac Network. We assume that the hardware of the machine will automatically take care of the boundary connection as desired. If the Illiac Network is being considered as a string of PEs, the following statement moves data in registers $R0$ from each PE to its upper neighbor:

$$R0[p] \leftarrow R0[p+1]$$

The routing strategy for a 2D-array is:

$R1(p_1+1, p_2) \leftarrow R1(p_1, p_2)$ (* up *)

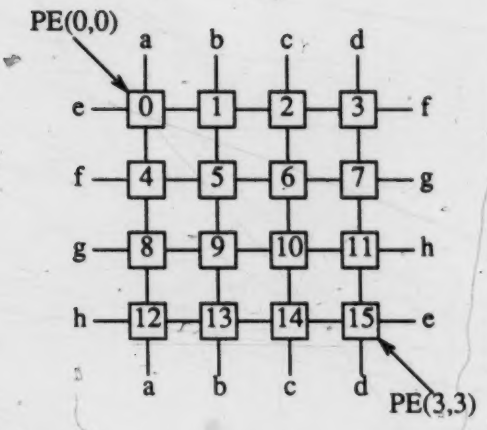
$R1(p_1-1, p_2) \leftarrow R1(p_1, p_2)$ (* down *)

$R1(p_1, p_2-1) \leftarrow R1(p_1, p_2)$ (* left *)

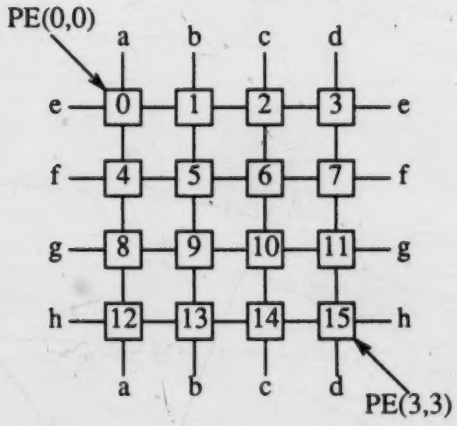
$R1(p_1, p_2+1) \leftarrow R1(p_1, p_2)$ (* right *)

3.5. CONCLUSION

In this chapter, we have described the machine model upon which parallel algorithms on relaxation labeling are developed. Notice that mesh-connected Illiac Network can be viewed as a ring, and the Torus Network as an array. So, in this thesis, we will actually developed parallel algorithms on 1D and 2D networks and analyze them according to time complexity and partition flexibility. Partition flexibility is studied in terms of the ease in partitioning of the task among few PEs. Time complexity is analyzed in terms of processing complexity and communication complexity.



(a)



(b)

Fig. 3.2 Examples of mesh.

(a) A 4×4 Illiac Network.

(b) A 4×4 Torus Network.

CHAPTER 4
PARALLEL ALGORITHMS AND PERFORMANCE ANALYSIS ON RELAXATION LABELING

4.1. INTRODUCTION

The time complexity required to solve a particular problem on a given SIMD machine can be expressed as a combination of the computation and the communication complexities. Optimal performance in terms of time complexity cannot be achieved by maximizing the degree of parallelism exploited in the computation and the communication components independently. Frequently tradeoffs have to be made between the two components to minimize the overall complexity subject to some given hardware constraints, such as limited number of PEs, registers, local memory capabilities, etc.

Based on the general computational model discussed in Chapter 2, parallel algorithms on one iteration of the relaxation labeling problem are developed on SIMD machines using mesh interconnection network. Each object is considered to have N neighbors. Since the indices of PEs start from 0, therefore, in order to simplify notations, object, label and matrix indices will also start from 0.

Parallel algorithms are first developed with $M \times N$ PEs. In reality, we are usually given a machine with number of PEs less than that required and hence the partitioning of data and computation among less PEs is required. Parallel algorithms with few PEs will be discussed. Parallel algorithms developed on mesh will be presented in section 4.2 and the performance analysis can be found in section 4.3. Conclusions of the chapter are presented in section 4.4.

4.2. PARALLEL ALGORITHMS ON SIMD MESH MACHINES

Here, we repeat the generalized relaxation formulation using Rosenfeld *et al.*'s non-linear updating rule. Given that matrix L is a matrix of size $M \times N$ and $R_{\lambda i}$'s are the compatibility matrices for different λ and i. The size of each $R_{\lambda i}$ is also $M \times N$. There are totally $M \times N$ $R_{\lambda i}$ matrices. Defining $g_5 = \sum_{\lambda''}$, the binary operation $\bar{g}_4 = \times$, $g_3 = \sum_{j}$, $g_2 = \sum_{\lambda'}$ and the binary operation $g_1 = \times$, the generalized formula for the updating rule is

$$G_3(L, R_{\lambda i}) = g_3 g_2 \bar{g}_1 (L^{(k)}, R_{\lambda i})$$

and

$$L[\lambda, i] = \frac{\bar{g}_4 \left[L[\lambda, i], G_3(L, R_{\lambda i}) \right]}{g_5 \bar{g}_4 \left[L[\lambda'', i], G_3(L, R_{\lambda'' i}) \right]}$$

4.2.1. Parallel Algorithms with MN PEs on Illiac Network

Consider a two-dimensional mesh of size M by N with an Illiac Network (Fig. 4.1). The transfer of data on the network can be along one of the two dimensions. Horizontally, when being stretched out, all PEs together form a ring. Vertically, the PEs in each column link into a ring. This architecture is a natural structure for the generalized relaxation labeling model. If the two-dimensional G_3 operation is performed in each PE, the two dimensions of the mesh can be used for data transfers along the remaining two dimensions specified in the general model. The L matrix is first superimposed onto the array of PEs such that PE(λ, i) contains $L[\lambda, i]$ together with the compatibility matrix $R_{\lambda i}$. Then the L elements are circulated horizontally. Operation $\bar{g}_1$ is performed in each PE after each horizontal transfer and the resulting

$\bar{g}_1(L[\lambda', j], R_{\lambda i}[\lambda', j])$ is stored into a matrix in PE(λ, i). Eventually, PE(λ, i) will contain a $M \times N$ matrix of $\bar{g}_1(L[\lambda', j], R_{\lambda i}[\lambda', j])$ so that functions g_2 and g_3 will be applied locally in each PE to each column of the matrix and to the resulting row respectively. The normalizing factor on each object can be obtained by circulating partial results in each PE vertically. Each PE(λ, i) will thus accumulate its own normalizing factor to update $L[\lambda, i]$. Fig. 4.2 shows the PE contents of a $M \times N$ Illiac mesh. Below is the procedure *IlliacG*$_3$*_MN* for one iteration of G_3 operation.

Procedure *IlliacG*$_3$*_MN* (*old*, *r*, *new*)

```
// Purpose:
//        To compute G3[R, L] on mesh with Illiac network using MN PEs.
// Input in PE(λ, i):
//        old  =  register containing L[λ, i].
//        r    =  M by N matrix, R_λi.
// Output in PE(λ, i):
//        new  =  register containing G3[R_λi, L].
// Others:
//        tmp  =  matrix of M by N for the storage of g1(R_λi, L)
//        c's  =  counters.

begin
    s := p1        //PE's row index//
    t := p2        //PE's column index//

    // Circulate the L[λ, i] horizontally and perform ḡ1 in each PE.
    // s, t are indices used to ensure that the results are stored in the right
    // positions of the matrix tmp to make ready for g2 and g3 operations.

    for c1 := 1 to M*N do
        tmp[s, t] := ḡ1(r[s, t], old)
        s := ( ⌊(t+1) / N⌋+s) mod M
        t := (t+1) mod N
        old(p1, p2−1) ← old(p1, p2)   // shift left //
    endfor //c1

    // Perform g2 on each column of matrix tmp and store results
```

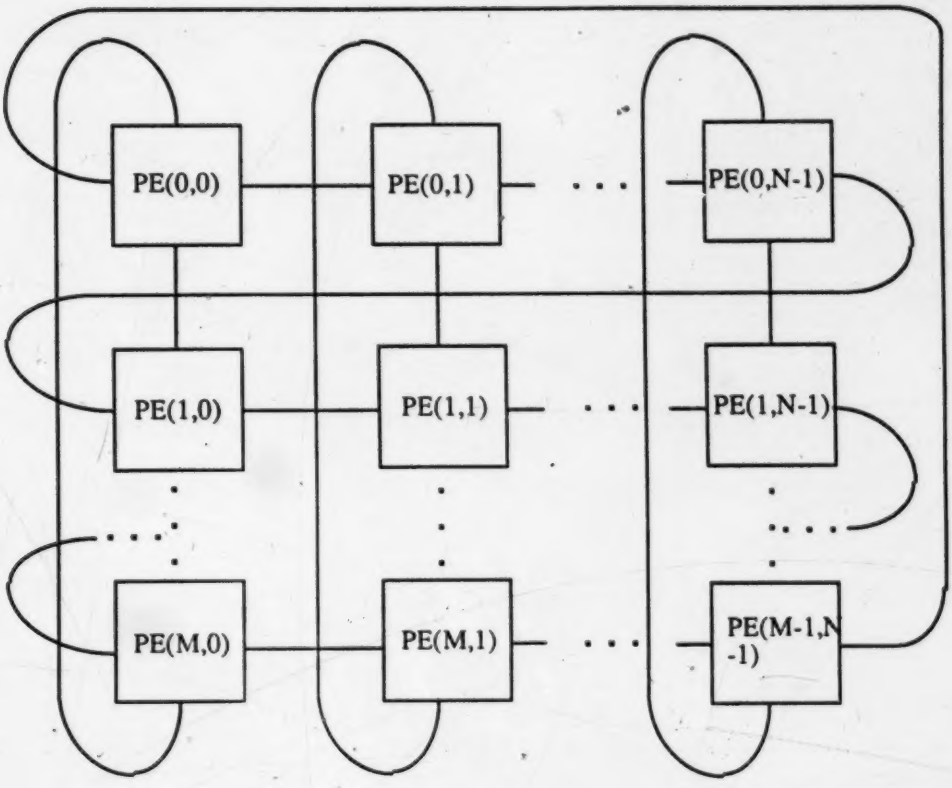


Fig. 4.1 A M by N Illiac-mesh.

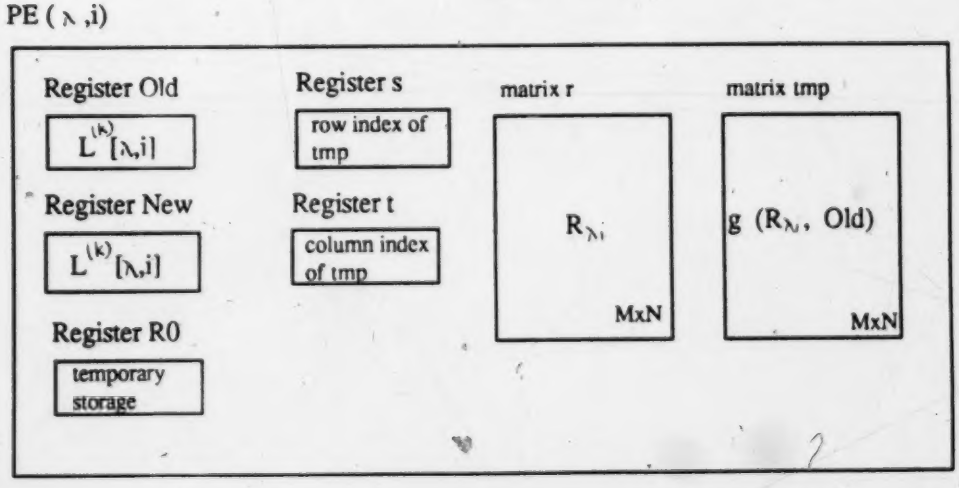


Fig. 4.2 Contents of PE (λ ,i).

```
    // into the 0-th row of tmp. Function g3 is performed on the
    // 0-th row with result stored in new.

    for c1 := 0 to N-1 do
        for c2 := 1 to M-1 do
            tmp [0, c1] := g2(tmp [0, c1], tmp [c2, c1])
        endfor //c2
        new := g3 (new, tmp [0, c1])
    endfor //c1

    new := tmp [0, 0]
end // IlliacG3_MN
```

Procedure *IlliacG*$_3$*_MN* can be called for computing one iteration update of the discrete, fuzzy, linear probabilistic models and the support function. The updated labeling, $L[\lambda, i]$ is stored in register *new* of PE(λ, i). The overall general algorithm for one iteration of the non-linear probabilistic relaxation labeling is provided in procedure *mesh_MN*.

Procedure *mesh_MN* (*old*, *r*, *new*)

```
// Purpose:
//          To compute one iteration to update L with MN PEs on mesh.
// Input in PE(λ, i):
//          old   =   register containing L[λ, i].
//          r     =   M by N matrix, R_λi.
// Output in PE(λ, i):
//          old   =   register containing updated L[λ, i].
// Others:
//          new   =   register used for accumulating the normalizing factor
//          R0    =   register used as temporary storage in an transfer.
//          c's   =   counters.

begin
    IlliacG3_MN (old, r, new)
    new := g4(new, old)

    // Register old store the numerator and register new is used
```

```
// to store the normalizing factor.

old := new

// Perform normalization along the vertical dimension of the PEs.
// Resulting normalizing factor is stored in register new of each PE.

for c1 := 1 to M-1 do
      R0(p1-1, p2) ← new(p1, p2)        (* shift up *)
      new := g5(R0, new)
endfor //c1

   old := old / new
end // mesh_MN
```

4.2.2. *Parallel Algorithms with MN PEs on Torus Network*

In procedure *IlliacG*$_3$*_MN*, all the results $g_1(L[\lambda', j], R_{\lambda i}[\lambda', j])$'s are collected into matrix *tmp* of PE(λ, i). The memory location for each element of *tmp* is determined from each PE's own address so that each $L[\lambda', j]$ will meet with the corresponding $R_{\lambda i}[\lambda', j]$ and is stored into the right location ready for g_2 and g_3 operations. The procedure requires the storage of both the R and *tmp* matrices in each PE. This section provides an alternative strategy, the use of skewed memory, on a Torus network so that only $R_{\lambda i}$ needs to be stored in each PE(λ, i).

The initial data distribution among the PEs is the same as that described in the previous section except that $R_{\lambda i}$ is stored in a skewed fashion in local memory of PE(λ, i). The transfer of each L element to each of the PEs is accomplished by a series of left and up movements. An appropriate g_i, i=1, 2, or 3, is performed after each transfer with partial updating result of $L[\lambda, i]$ in PE(λ, i). Since the Torus and Illiac networks differ only in the horizontal connections of PEs and the accumulation of the normalization factor is along the vertical dimension of the mesh, therefore the same normalization algorithm will be used by both networks.

Now, let us consider the placements of the elements of matrix $R_{\lambda i}$ in the local memory of each PE. At the beginning of each iteration, $L[\lambda, i]$ is stored in PE(λ, i). The $L[\lambda, i]$ will then circulate vertically up $(M-1)$ steps, shift left one step, and the two procedures are repeated until every L element has visited every PE. Function g_1 is applied to every $R_{\lambda i}[\lambda', j]$ and $L[\lambda', j]$ pair. If the index of an element in $R_{\lambda i}$ is $[x, y]$, $0 \le x \le M-1$ and $0 \le y \le N-1$, then it will be stored in $r[x', y']$ of PE(λ, i) where

$$x' = \left[x - \lambda - [(y-i) \bmod N](M-1)\right] \bmod M,$$

$$y' = (y-i) \bmod N,$$

and λ is the row index of the PE. Fig. 4.3 gives an example on the storage of R's in a 4×3 Torus mesh. Below is procedure *TorusG*$_3$*_MN*.

Procedure *TorusG*$_3$*_MN* (*old*, *r*, *new*)

```
// Purpose:
//        To compute G3[R, L] on mesh with Torus network using MN PEs.
// Input in PE(λ, i):
//        old  =  register containing L[λ, i].
//        r    =  M by N matrix, Rλi.
// Output in PE(λ, i):
//        new  =  register containing G3[Rλi, L].
// Others variable in PE(λ, i):
//        save =  register used to store original L[λ, i]
//        tmp  =  register used to store the temporary result.
//        c's  =  counters.

begin
    initialize new          //according to different models//
    save := old

    for c1 = 0 to N-1 do
        tmp := g1(r[0, c1], old)

        // This loop is to shift the l's vertically in a round ribon
        // fashion so that g2 is computed for every column.
```

PE(0,0)	PE(0,1)	PE(0,2)
$\hat{R}_{00}[0,0]$ $\hat{R}_{00}[3,1]$ $\hat{R}_{00}[2,2]$ $\hat{R}_{00}[1,0]$ $\hat{R}_{00}[0,1]$ $\hat{R}_{00}[3,2]$ $\hat{R}_{00}[2,0]$ $\hat{R}_{00}[1,1]$ $\hat{R}_{00}[0,2]$ $\hat{R}_{00}[3,0]$ $\hat{R}_{00}[2,1]$ $\hat{R}_{00}[1,2]$	$\hat{R}_{01}[0,1]$ $\hat{R}_{01}[3,2]$ $\hat{R}_{01}[2,0]$ $\hat{R}_{01}[1,1]$ $\hat{R}_{01}[0,2]$ $\hat{R}_{01}[3,0]$ $\hat{R}_{01}[2,1]$ $\hat{R}_{01}[1,2]$ $\hat{R}_{01}[0,0]$ $\hat{R}_{01}[3,1]$ $\hat{R}_{01}[2,2]$ $\hat{R}_{01}[1,0]$	$\hat{R}_{02}[0,2]$ $\hat{R}_{02}[3,0]$ $\hat{R}_{02}[2,1]$ $\hat{R}_{02}[1,2]$ $\hat{R}_{02}[0,0]$ $\hat{R}_{02}[3,1]$ $\hat{R}_{02}[2,2]$ $\hat{R}_{02}[1,0]$ $\hat{R}_{02}[0,1]$ $\hat{R}_{02}[3,2]$ $\hat{R}_{02}[2,0]$ $\hat{R}_{02}[1,1]$
PE(1,0)	PE(1,1)	PE(1,2)
$\hat{R}_{10}[1,0]$ $\hat{R}_{10}[0,1]$ $\hat{R}_{10}[3\,2]$ $\hat{R}_{10}[2,0]$ $\hat{R}_{10}[1,1]$ $\hat{R}_{10}[0,2]$ $\hat{R}_{10}[3,0]$ $\hat{R}_{10}[2,1]$ $\hat{R}_{10}[1,2]$ $\hat{R}_{10}[0,0]$ $\hat{R}_{10}[3,1]$ $\hat{R}_{10}[2,2]$	$\hat{R}_{11}[1,1]$ $\hat{R}_{11}[0,2]$ $\hat{R}_{11}[3,0]$ $\hat{R}_{11}[2,1]$ $\hat{R}_{11}[1,2]$ $\hat{R}_{11}[0,0]$ $\hat{R}_{11}[3,1]$ $\hat{R}_{11}[2,2]$ $\hat{R}_{11}[1,0]$ $\hat{R}_{11}[0,1]$ $\hat{R}_{11}[3,2]$ $\hat{R}_{11}[2,0]$	$\hat{R}_{12}[1,2]$ $\hat{R}_{12}[0,0]$ $\hat{R}_{12}[3,1]$ $\hat{R}_{12}[2,2]$ $\hat{R}_{12}[1,0]$ $\hat{R}_{12}[0,1]$ $\hat{R}_{12}[3,2]$ $\hat{R}_{12}[2,0]$ $\hat{R}_{12}[1,1]$ $\hat{R}_{12}[0,2]$ $\hat{R}_{12}[3,0]$ $\hat{R}_{12}[2,1]$
PE(2,0)	PE(2,1)	PE(2,2)
$\hat{R}_{20}[2,0]$ $\hat{R}_{20}[1,1]$ $\hat{R}_{20}[0\,2]$ $\hat{R}_{20}[3,0]$ $\hat{R}_{20}[2,1]$ $\hat{R}_{20}[1,2]$ $\hat{R}_{20}[0,0]$ $\hat{R}_{20}[3,1]$ $\hat{R}_{20}[2,2]$ $\hat{R}_{20}[1,0]$ $\hat{R}_{20}[0,1]$ $\hat{R}_{20}[3,2]$	$\hat{R}_{21}[2,1]$ $\hat{R}_{21}[1,2]$ $\hat{R}_{21}[0,0]$ $\hat{R}_{21}[3,1]$ $\hat{R}_{21}[2,2]$ $\hat{R}_{21}[1,0]$ $\hat{R}_{21}[0,1]$ $\hat{R}_{21}[3,2]$ $\hat{R}_{21}[2,0]$ $\hat{R}_{21}[1,1]$ $\hat{R}_{21}[0,2]$ $\hat{R}_{21}[3,0]$	$\hat{R}_{22}[2,2]$ $\hat{R}_{22}[1,0]$ $\hat{R}_{22}[0,1]$ $\hat{R}_{22}[3,2]$ $\hat{R}_{22}[2,0]$ $\hat{R}_{22}[1,1]$ $\hat{R}_{22}[0,2]$ $\hat{R}_{22}[3,0]$ $\hat{R}_{22}[2,1]$ $\hat{R}_{22}[1,2]$ $\hat{R}_{22}[0,0]$ $\hat{R}_{22}[3,1]$
PE(3,0)	PE(3,1)	PE(3,2)
$\hat{R}_{30}[3,0]$ $\hat{R}_{30}[2,1]$ $\hat{R}_{30}[1\,2]$ $\hat{R}_{30}[0,0]$ $\hat{R}_{30}[3,1]$ $\hat{R}_{30}[2,2]$ $\hat{R}_{30}[1,0]$ $\hat{R}_{30}[0,1]$ $\hat{R}_{30}[3,2]$ $\hat{R}_{30}[2,0]$ $\hat{R}_{30}[1,1]$ $\hat{R}_{30}[0,2]$	$\hat{R}_{31}[3,1]$ $\hat{R}_{31}[2,2]$ $\hat{R}_{31}[1,0]$ $\hat{R}_{31}[0,1]$ $\hat{R}_{31}[3,2]$ $\hat{R}_{31}[2,0]$ $\hat{R}_{31}[1,1]$ $\hat{R}_{31}[0,2]$ $\hat{R}_{31}[3,0]$ $\hat{R}_{31}[2,1]$ $\hat{R}_{31}[1,2]$ $\hat{R}_{31}[0,0]$	$\hat{R}_{32}[3,2]$ $\hat{R}_{32}[2,0]$ $\hat{R}_{32}[1,1]$ $\hat{R}_{32}[0,2]$ $\hat{R}_{32}[3,0]$ $\hat{R}_{32}[2,1]$ $\hat{R}_{32}[1,2]$ $\hat{R}_{32}[0,0]$ $\hat{R}_{32}[3,1]$ $\hat{R}_{32}[2,2]$ $\hat{R}_{32}[1,0]$ $\hat{R}_{32}[0,1]$

Fig. 4.3. Storage of R in a 4×3 Torus mesh.

```
        for c2 := 1 to M−1 do
            old(p1−1, p2) ← old(p1, p2)                    // shift up//
            tmp := g2(tmp, g1(old, r[c2, c1])
        endfor //c2

        // g3 is applied once after g2 has been applied on its M operands.

        new := g3(new, tmp)
        old(p1, p2−1) ← old(p1, p2)                        //shift left
    endfor //c1
    old := save

end //TorusG3_MN
```

4.2.3. *Parallel Algorithms with Less PEs*

Given a SIMD machine, the number of PEs is fixed and all PEs execute the same instructions broadcasted from the CU and collaborately, they tackle the same task at a time. Therefore, we are not concerned with throughput in individual PE but rather with strategies which best utilize the given hardware to optimize the overall time complexity in solving the problem. We thus contemplate on maximizing the overall time complexity with less PEs instead of finding the number of PEs which optimizes the throughput/utilization in each PE.

Consider a mesh with XY PEs, $X \le M$ and $Y \le N$. Define two factors,

$$F_1 = \left\lceil \frac{M}{X} \right\rceil \quad \text{and} \quad F_2 = \left\lceil \frac{N}{Y} \right\rceil.$$

Two storage schemes can be employed to partition the L matrix among the PEs. In *consecutive storage*, the matrix is divided into small consecutive blocks each being of dimension $F_1 \times F_2$. Element $L[\lambda, i]$ is stored in PE(p_1, p_2) with

$$p_1 = \left\lfloor \frac{\lambda}{F_1} \right\rfloor \quad \text{and} \quad p_2 = \left\lfloor \frac{i}{F_2} \right\rfloor$$

The PEs near the right or bottom boundary may contain no data if $F_1X - M > F_1$ or $F_2Y - N > F_2$ respectively. The second scheme uses *cyclic storage* in which all elements $L[\lambda, i]$ satisfying relations $p_1 = \lambda \bmod X$ and $p_2 = i \bmod Y$ are stored in PE(p_1, p_2). Extra fill-ins are required by both schemes if M and N are not divisible by X and Y respectively. An example of the consecutive and cyclic storage schemes are illustrated in Fig. 4.4.

Algorithms with consecutive scheme change only in granularity of operations. That is, an linear algebra computation operation on a single element is replaced by having the same operations on a submatrix of size $F_1 \times F_2$. In the cyclic storage scheme, the granularity is the same as the original storage scheme. The submatrices can be viewed as a group of subtask with each task being indentical to the original task but is much small in size. In [Joh87], the subtasks are described as storage planes that are brought to the processing plane (Fig. 4.5).

With fewer PEs, each PE is responsible of updating a submatrix of $F_1 \times F_2$ L elements. Stored in each PE is also a subdivided compatibility matrix organized into a four dimensional array, $F_1 \times F_2 \times F_1X \times F_2Y$. Procedures *IlliacG*$_3$*_XY* and *mesh_XY* use consecutive storage scheme are shown below.

Procedure *IlliacG*$_3$*_XY* (*old*, *r*, *new*)

```
// Purpose:
//     To compute G3[R, L] with XY PEs on mesh.
// Input in each PE(λ, i):
//     old   =   array of size [F1,F2] containing L[λ, i]'s.
//     r     =   matrix of dimension [F1, F2][M, N]
// Output in each PE(λ, i):
//     new   =   array of size [F1,F2] containing G3(Rλi, L[λ,i]).
// Other variables in each PE:
//     s, t  =   registers used for indexing.
```

00	01	02	03	04	05	06
10	11	12	13	14	15	16
20	21	22	23	24	25	26

(a)

PE(0,0)				PE(0,1)			
00	01	02	03	04	05	06	*
10	11	12	13	14	15	16	*
PE(1,0)				PE(1,1)			
20	21	22	23	24	25	26	*
*	*	*	*	*	*	*	*

(b)

PE(0,0)				PE(0,1)			
00	02	04	06	01	03	05	*
20	22	24	26	21	23	25	*
PE(1,0)				PE(1,1)			
10	12	14	16	11	13	15	*
*	*	*	*	*	*	*	*

(c)

* -- *Fill-in for matrix in a PE.*

Fig 4.4 Consecutive and cyclic storage of a matrix with $M=3, N=7, X=Y=2$.
(a) Original unpartitioned matrix.
(b) Consecutive storage.
(c) Cyclic storage.

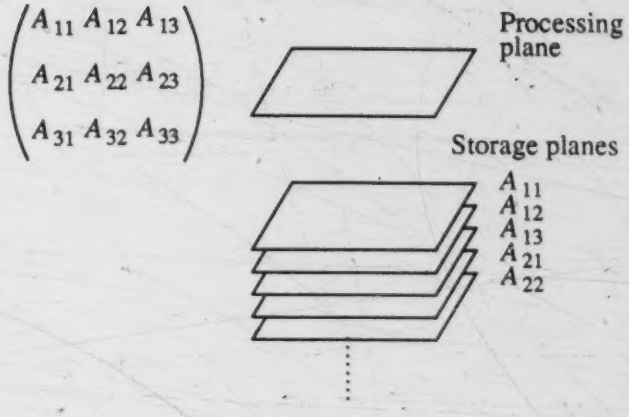


Fig. 4.5. Processing and storage planes.

```
//    c's   =   counters.
//    tmp   =   array of size [F1, F2][M, N] used to store results after g1.

begin
   s := row index of PE * F1
   t := column index of PE * F2

   for c1 := 1 to X*Y do

      // Each PE contains F1×F2 L elements

      for c2 := s to s+F1−1 do
         for c3 := t to t+F2−1 do

            // For each L element, perform g1 on F1×F2 R for
            // the update of F1×F2 L elements.

            for c4 := 0 to F1−1 do
                  for c5 := 0 to F2−1 do
                     tmp[c4,c5][c2,c3] := g1(r[c4,c5][c2,c3],old[c4,c5])
```

```
                endfor //c5
            endfor //c4

        endfor //c3
    endfor //c2
    s := ⌊⌊(t+F2) / N⌋ * F1+s⌋ mod M
    t := (t+F2) mod N

    // Transfer F1×F2 L elements from neighbor.

    for c2 := 0 to F1−1 do
        for c3 := 0 to F2−1 do
            old[c2, c3](p1, p2−1) ← old[c2, c3](p1, p2)
        endfor //c3
    endfor //c2

endfor //c1

// To update F1×F2 L elements,

for c1 := 0 to F1−1do
    for c2 := 0 to F2−1 do

        Initialize new[c1, c2]

        // we consider only the first M×N elements stored in tmp
        // and neglect the fill-ins.

        for c3 := 0 to N−1 do
            R0 := tmp[c1, c2][0, c3]
            for c4 := 1 to M−1 do
                R0 := g2(R0, tmp[c1, c2][c3, c4])
            endfor //c4
            new[c1, c2] := g3 (new[c1, c2],R0)
        endfor //c3

    endfor //c2
endfor //c1
end (* IlliacG3_XY *)

Procedure mesh_XY (old, r, new)
```

```
// Purpose:
//      To compute one iteration to update L with MN PEs on mesh.
// Input in PE(λ, i):
//      old   =    array of size [F1, F2] containing L[λ, i]'s.
//      r     =      M by N matrix, Rλi.
// Output in PE(λ, i):
//      old   =    array of size F1, F2] containing updated L[λ, i]'s.
// Other variables in each PE:
//      new   =    array of size F1, F2] containing the numerators for normalization.
//      R0    =    register for accumulating the normalizing factor.
//      R1    =    register for temporary storage in a transfer.
//      c's   =    counters.

begin
   IlliacG3_XY (old, r, new)

   for c1 := 0 to F1−1 do
      for c2 := 0 to F2−1 do
         new[c1, c2] := g4(new[c1, c2], old[c1, c2])
      enfor //c2
   endfor //c1

   for c1 := 0 to (F2−1) do
   .  R0 := new[0, c1]

      // Perform g5 on the columns of matrix new in each PE.

      for c2 := 1 to F1−1 do
         R0 := g5(R0, new[c2, c1])
      endfor //c2

      // Circulate partial results vertically.  The resulting normalizing factor
      // will be in R0 of each PE.

      for c2 := 1 to X−1 do
         R1(p1−1, p2) ← R0(p1, p2)
         R0 := g5(R0, R1)
      endfor //c2

      for c2 := 1 to F1−1 do
         old[c2, c1] := new[c2, c1]/R0
      endfor //c2
```

endfor //c_1
end (* *mesh_XY* *)

4.3. PERFORMANCE ANALYSIS

Displayed in Table 4.1 are the performance of the mesh algorithms. Recorded in the Table are the number of steps required for each g function, other operations including multiplications/divisions (*mult.*) and additions (*add*), the number of transfer steps used, communication, computation and overall time complexities of each approach using different number of PEs.

Using MN PEs, both the Torus and Illiac approaches yield an overall time complexity of $O(MN)$. However, the Torus approach is more advantageous in that it uses less memory space and does not induce any overhead on index management. The skewed memory strategy imposed overhead only on loading the R matrices into the local memory of the PEs, but not in each iteration of relaxation operations. Indeed the skewed memory approach can also be used on the Illiac network provided that g_2 and g_3 are the same. Notice that the decrease in the number of PEs increases only the computation complexity. The communication complexity is unaffected because every L element has to visit every PE. So the MN steps employed in transferring MN L elements is unavoidable. With less PEs, the increase in computation complexity is inversely proportional to the number of PEs used.

4.4. CONCLUSION

The time complexity of relaxation labeling on a sequential computer is $O(M^2N^2)$ per iteration. With M^2N^2 PEs connected by a mesh interconnection network, the computational complexity is reduced to $O(M+N)$ with communication complexity being at least $O(MN+M)$ which gives an overall time complexity of at least

Table 4.1. Performance of the mesh algorithms.

	MN PEs		XY PEs
	(Illiac)	(Torus)	
g_1	MN	MN	$XYF_1^2F_2^2$
g_2	$N(M-1)$	$N(M-1)$	$(M-1)NF_1F_2$
g_3	N	N	F_1F_2N
T_{G_3}	MN	MN	XYF_1F_2
Other (G_3)	$3MN$ (*mult.*+*add*)	—	$3XYF_1F_2$*add*+$(4XYF_1F_2+2)$*mult*
g_4	1	1	F_1F_2
g_5	$M-1$	$M-1$	$F_1F_2+XF_2$
T	$M-1$	$M-1$	XF_1
Other	1 *mult.*	1*mult.*	F_1F_2 *mult.*
Comm.	$O(MN)$	$O(MN)$	$O(MN)$
Comp.	$O(MN)$	$O(MN)$	$O(MNF_1F_2)$ $=O\left(\frac{M^2N^2}{XY}\right)$
Overall	$O(MN)$	$O(MN)$	$O\left(\frac{M^2N^2}{XY}\right)$

$O(MN+M)$. Using only MN PEs connected by a Illiac or Torus mesh network, the overall time complexity is also $O(MN)$. Extra storage is required for data alignment if the Illiac-mesh network is employed. Parallel algorithms on the Torus network have the advantage of requiring less local memory in each PE by using skewed memory strategy. However, as in some approaches, if the g_2 and g_3 operations are identical, both algorithms can employ the same amount of local memory with the

Illiac algorithms being slightly modified. Both algorithms can easily be modified to work on fewer PEs.

CHAPTER 5
CONCLUSION

5.1. CONTRIBUTION OF THE THESIS

In this thesis, we have derived a four-dimensional general computational model on relaxation labeling upon which several parallel algorithms on SIMD computers are developed. The general model includes the four models by Rosenfeld *et al.*, the statistical formulation by Peleg, the arithmetic approach by Zucker *et al.*, and the support function by Hummel *et al.* A table listing the different notations and terminologies used by the researchers in the original formulations of the models is included. Parallel algorithms are written on Illiac-mesh and Torus-mesh computers using different number of PEs.

In the general computation model, a labeling is defined as a $M \times N$ matrix and a compatibility matrix of the same dimension is defined on each object-label pair. The general model involves four dimensions of computations denoted by j, λ', i, λ which represents the axes of an object's neighbors, a neighbor's possible labels, different objects and an object's possible labels, respectively. The update of a $L^{(k)}[\lambda, i]$ is a mapping of a 3D-array of compatibility matrices, $R_{\lambda i}$'s, with the compatibility matrix, L, to give a scalar, the updated $L^{(k+1)}[\lambda, i]$. The model can be generalized to handle J neighbors and W possible labels for each object. Five functions are defined on the general model for updating the labeling matrix in each iteration. Three of the functions are multi-operand operations.

The time complexity of the original relaxation labeling problem is $O(M^2N^2)$ per iteration on a sequential computer. Several parallel algorithms are developed which greatly reduced the time complexity. Parallel algorithms using MN PEs yield a time

complexity of also $O(MN)$ on Illiac-mesh and Torus-mesh. The algorithms on the Torus network has the advantage of using less local memory space in each PE. Modifications are made on Illiac algorithms to use less PEs which yields an overall time complexity of $O(\frac{M^2N^2}{Z})$ where Z is the number of PEs used.

5.2. FURTHER WORK

In this thesis, we studied a generalized computational model and developed parallel algorithms on relaxation labeling. Several commonly used approaches are included in the generalized model. Further generalization on the model can be done by including more models and study their similarities and differences in terms of computations. Further generalization on the algorithms can be done by developing parallel algorithms on other interconnection networks. A more thorough study of time complexity analysis can be made possible if parallel machines are available or with the aid of simulations on a sequential machine.

Five functions are defined in the general models among which three are multi-operand operations. With today's technology, computers are handling multi-operand operations using binary operators. Hence, a potential speedup in relaxation labeling is to develop operators which can execute multi-operand instructions with shorter time.

Communications between PEs are also an expensive overhead. For instance, with mesh, even if the number of PEs is greater than MN, the communication complexity is at least $O(MN)$ leaving the overall time complexity to be at least $O(MN)$. Special parallel architectures can be developed which facilitate data transmission along all different dimension.

References

[BaT76] H. G. Barrow and J. M. Tenenbaum, MYSYS: A system for reasoning about scenes, SRI AI Tech. Rep. 121, Stanford Res. Inst., Menlo Park, CA, 1976.

[DaR80] A. J. Danker and A. Rosenfeld, Strip detection using relaxation, *Pattern Recognition 12*, (1980), 97-100.

[DaR77] L. S. Davis and A. Rosenfeld, Curve segmentation by relaxation labeling, *IEEE Trans. Comput. 26*, (1977), 1053-1057.

[Dav79] L. S. Davis, Shape matching using relaxation techniques, *IEEE Trans. PAMI 1*, (Jan. 1979), 60-72.

[DNS81] E. Dekel, D. Nassimi and S. Sahni, Parallel matrix and graph algorithms, *SIAM J. Comp. 10*, 4 (1981), 657-675.

[DiM87] V. Dixit and D. I. Moldovan, Semantic network array processor and its applications to image understanding, *IEEE PAMI 9*, 1 (Jan. 1987), 153-160.

[EYR78] J. Eklundh, H. Yamamoto and A. Rosenfeld, Relaxation methods in multispectral pixel classifications, 662, Comput. Sci. Center, Univ. Maryland, Jul. 1978.

[FLN85] Z. Fang, X. Li and L. Ni, Parallel Algorithms for image template matching on hypercube SIMD computers, *IEEE CAPIDM Workshop*, 1985, 33-40.

[FaB81] O. Faugeras and M. Berthod, Improving consistency and reducing ambiguity in stocastic labeling: An optimization approach, *IEEE Trans. PAMI 3*, 4 (Jul. 1981), 412-424.

[FER81] G. Fekete, J. Eklundh and A. Rosenfeld, Relaxation: Evaluation and applications, *IEEE PAMI 3*, 4 (Jul. 1981), 460-469.

[GW87] J. Gu, W. Wong and T. Henderson, A parallel architecture for discrete relaxation algorithm, *IEEE Trans PAMI 9*, 6 (Nov. 1987), 816-831.

[HaR78] Scene analysis, homomorphism and arrangements, *Machine Vision Workshop*, New York, 1978.

[HaS79] R. Haralick and L. Shapiro, The consistent labeling problem: part I, *IEEE Trans. PAMI 1*, 2 (Apr. 1979), 173-184.

[Hay78] Hayes, Reading handwritten works using hierarchical relaxation, 623, Ph.D. dissertation, Dept. Comput. Sci. Center, Univ. Maryland, Jan. 1978.

[Hin79] G.E. Hinton, *Relaxation and its role in vision*, Ph.D. dissertation, Univ. Edinburgh, Dec. 1979.

[HuZ83] R. A. Hummel and S. W. Zucker, On the foundations of relaxation labeling processes, *IEEE Trans. PAMI 5*, 3 (May 1983), 267-287.

[HwB84] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, 1984.

[Joh87] S. L. Johnsson, Communication efficient basic linear algebra computations on hypercube architectures, *J. of Parallel and Dist. Comput. 4*, 2 (1987), 133-172.

[KTM88] M. Kamada, K. Toraichi, R. Mori, K. Yamamoto and H. Yamada, A parallel architecture for relaxation operations, *Pattern Recognition*, 1988, 175-181.

[LZR78] A. Lev, S. Zucker and A. Rosenfeld, Determining compatibility coefficients for curve enhancement relaxation process, *IEEE Trans. SMC*

8, (Jul. 1978), 548-555.

[LYD88] H. Liu, T. Yong and A. Das, A multilevel parallel processing approach to scene labeling problems, *IEEE Trans. PAMI 10* , 4 (Jul. 1988), 586-590.

[PeR78] S. Peleg and A. Rosenfeld, Straight edge enhancement and mapping, *Comput. Sci. Center, Univ. Maryland*, Sept. 1978.

[PeR79] S. Peleg and A. Rosenfeld, Breaking substitution ciphers using a relaxation algorithm, *Commun. Ass. Comput. Mach. 22*, (Dec. 1979), 598-605.

[Pel79] S. Peleg, Ambiguity reduction in handwriting with ambiguous segmentation and uncertain interpretation, *Comput. graphics Image Processing*, 19979, 235-245.

[Pel80] S. Peleg, A new probabilistic relaxation scheme, *IEEE Trans PAMI 2*, 4 (Jul. 1980), 362-369.

[Pra80] J. M. Prager, Extracting and labeling boundary segments in natural scenes, *IEEE Trans. PAMI 2*, 1 (Jan. 1980), 16-27.

[ReK85] D. Reisis and V.K. Kumar, Parallel processing of the labeling problem, *Proceedings of IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Nov. 1985, 381-385.

[RLS81a] J. A. Richards, D. A. Landgrebe and P. H. Swain, Pixel labeling by supervised probabilistic relaxation, *IEEE Trans. PAMI 3*, 2 (Mar. 1981), 188-191.

[RLS81b] J. A. Richards, D. A. Landgrebe and P. H. Swain, On the accuracy of Pixel relaxation labeling, *IEEE Trans. SMC 11*, 4 (Apr. 1981), 303-309.

[RHZ76] A. Rosenfeld, R. A. Hummel and S. W. Zucker, Scene labeling by relaxation operations, *IEEE Trans. SMC 6*, 6 (Jun. 1976), 420-433.

[RPR81] W. S. Rutkowski, S. Peleg and Azriel Rosenfeld, Shape segmentation using relaxation, *IEEE Trans. PAMI 3*, 4 (Jul. 1981), 368-375.

[SLZ77] B. Schachter, A. Lev, S. W. Zucker and A. Rosenfeld, An application of relaxation methods to edge reinforcement, *IEEE Trans. SMC 7*, (1977), 813-816.

[ThS86] M. Thathachar and P. S. Sastry, Relaxation labeling with learning automata, *IEEE Trans. PAMI 8*, 2 (Mar. 1986), 256-268.

[Van76] G. Vanderbrug, Experiments in iterative enhancement of linear features, *LARS Symp. Proc. Machine Process. Remote Sensed Data*, Purdue Univ., Laffayette, IN , Jun. 29-Jul. 1, 1976.

[Wal75] D. Waltz, Understanding line drawings of scenes with shadows, in *The Psychology of Computer Vision*, P. Winston (ed.), New York: McGraw-Hill, 1975.

[Zuc76] S. W. Zucker, Relaxation labeling and the reduction of local ambiguities, *Proc. 3rd. Int. Joint Conf. Pattern Recognition*, San Diego, CA, Nov. 1976, 852-861.

[ZuH77] S. W. Zucker and R. A. Hummel, Computing the shape of dot clusters, I: Labeling edge, interior and noise points, 543, May 1977.

[ZHR77] S. W. Zucker, R. A. Hummel and A. Rosenfeld, An application of relaxation labeling to line and cure enhancement, *IEEE Trans. Comput. 26*, (1977), 394-403, 922-929.

[ZKH78] S. Zucker, E. Krishnamurthy and R. Haar, Relaxation processes for scene labeling: Convergence, speed and stability, *IEEE Trans. SMC 8*, (Jan. 1978), 41-48.